Names and Binding in Type Theory

Ulrich Schöpp

Doctor of Philosophy
Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh
2006

Abstract

Names and name-binding are useful concepts in the theory and practice of formal systems. In this thesis we study them in the context of dependent type theory. We propose a novel dependent type theory with primitives for the explicit handling of names. As the main application, we consider programming and reasoning with abstract syntax involving variable binders.

Gabbay and Pitts have shown that Fraenkel Mostowski (FM) set theory models a notion of name using which informal work with names and binding can be made precise. They have given a number of useful constructions for working with names and binding, such as a syntax-independent notion of freshness that formalises when two objects do not share names, and a freshness quantifier that simplifies working with names and binding. Related to FM set theory, a number of systems for working with names have been given, among them are the first-order Nominal Logic, the higher-order logic FM-HOL, the Theory of Contexts as well as the programming language FreshML.

In this thesis we study how dependent type theory can be extended with primitives for working with names and binding. Our choice of primitives is different from that in FM set theory. In FM set theory the fundamental primitive for working with names is swapping. All other concepts such as α-equivalence classes and binding are constructed from it. For dependent type theory, however, this approach of constructing everything from swapping is not ideal, since it requires us to make strong assumptions on the type theory. For instance, the construction of α-equivalence classes from swapping appears to require quotient types. Our approach is to treat constructions such as α-equivalence classes and name-binding directly, turning them into primitives of the type theory. To do this, it is convenient to take freshness rather than swapping as the fundamental primitive.

Based on the close correspondence between type theories and categories, we approach the design of the dependent type theory by studying the categorical structure underlying FM set theory. We start from a monoidal structure capturing freshness. By analogy with the definition of simple dependent sums Σ and products Π from the cartesian product, we define monoidal dependent sums Σ^* and products Π^* from the monoidal structure. For the type of names $\mathbf{N}$, we have an isomorphism $\Sigma^*_{\mathbf{N}} \cong \Pi^*_{\mathbf{N}}$ generalising the freshness quantifier. We show that this structure includes α-equivalence classes, name binding, unique choice of fresh names as well as the freshness quantifier. In addition to the set theoretic model corresponding to FM set theory, we also give a realizability model of this structure.

The semantic structure leads us to a bunched type theory having both a dependent additive context structure and a non-dependent multiplicative context structure. This type theory generalises the simply-typed $\alpha\lambda$-calculus of O'Hearn and Pym in the additive direction. It includes novel monoidal products Π^* and sums Σ^* as well as hidden-name types H for working with names and binding.

We give examples for the use of the type theory for programming and reasoning with abstract syntax involving binders. We show that abstract syntax can be handled both in the style of FM set theory and in the style of Weak Higher Order Abstract Syntax. Moreover, these two styles of working with abstract syntax can be mixed, which has interesting applications such as the derivation of a term for the unique choice of new names.

Acknowledgements

First and foremost I would like to thank my supervisor Ian Stark for his great guidance, support and encouragement over the last few years.

I would also like to thank my second supervisor Alex Simpson, whose comments and questions helped to improve the contents of this thesis.

Having had the privilege of being at the Laboratory for the Foundations of Computer Science, I had the chance to learn a lot from many people, and I hope I have not wasted too many opportunities to do so. The Theory Postgraduate Courses were very helpful for learning about a range of topics. I learnt about category theory from Daniele Turi, about type theory from Randy Pollack, about interactive theorem proving from Martin Hofmann, about semantics from John Longley and Alex Simpson, to mention just the courses that are directly relevant to this thesis (I've enjoyed all the others very much as well, but there are too many to list them all). The administrative staff at LFCS were also very helpful and effective.

I had discussions with and benefited from the advice of James Cheney, John Longley, Maria Emilia Maietti, Marino Miculan, Gordon Plotkin, John Power and Randy Pollack.

I am grateful to Andrew Pitts and Gordon Plotkin for their examination of this thesis.

Thanks are also due to Luke Ong, who made it possible for me to follow Ian to Oxford for three months in 2003. I would like to thank the members of the Comlab at Oxford for their kind hospitality.

Last but not least, I'd like to say cheers to Anne Benoit, Catherine Canevet, Jennifer Tenzer, Tom Rigde, Dan Sheridan and Miki Tanaka, who made sure I got to the pub from time to time.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ulrich Schöpp*)

I have made this longer, because I have not had the time to make it shorter.

— BLAISE PASCAL, Lettres provinciales, letter 16, 1657

Table of Contents

Chapter 1

Introduction

Names are an everyday tool in the theory and practice of formal systems. They appear as variable names in the abstract syntax of programming languages, the π-calculus models computation by name-passing, references in ML are names of memory locations, to mention just a few examples. In these examples, names are used as abstract place-holders and have very little structure of their own. All we know about them is that they have an identity, i.e. we can compare them for equality, and that there are enough of them, i.e. we can always find an unused one. Despite, or perhaps because of, the fact that such names have so little structure, they have been found to be genuinely useful for working informally with formal systems. On the other hand, the formalisation of informal arguments with names has turned out to be surprisingly complicated.

Nevertheless, being able to work formally with names is very desirable. Of particular practical importance is the mechanisation of abstract syntax. For an example, consider the development of programming languages. It is dominated by the verification of standard properties, often by straightforward proofs. For instance, when designing a type system for a functional programming language one usually wants to verify standard properties such as subject-reduction. The proofs of such properties are usually routine verifications. However, since programming languages are often quite large, verifying such simple properties can be tedious and lengthy. Moreover the proofs are, due to their size, often hard to read and it is hard to be completely convinced of their correctness. This makes the theory of such formal systems a prime target for formal proof and proof automation. However, for the formalisation of proofs about programming languages it is necessary to formally represent the abstract syntax of the programming language. The treatment of names and name-binding in the abstract syntax presents a major obstacle in the formalisation effort. The problem is not that it is intrinsically difficult to formalise informal developments, but that formalisations require substantial additional effort. The goal of building formal systems with names is to bring formal work with names as close as possible to informal work, in order to make formalisations more convenient.

The topic of this thesis is the design of one such formal system, a dependent type theory with names. We propose a novel dependent type theory containing constructs for working with names and binding. As the semantic basis for this type theory we develop a categorical formulation of name binding. In this formulation, we characterise concepts such as α-equivalence classes and binding directly by means of universal constructions. We derive the primitives for working with names and binding in the type theory from this categorical structure, and we use bunches to build them into dependent type theory. As the main application of the type theory, we consider programming and reasoning with abstract syntax.

In this thesis we concentrate on the use of names for the formalisation of abstract syntax. Due to the practical importance of formalising abstract syntax, many other approaches have been proposed. We review the most important of them at the end of this chapter, but until then we focus entirely on the use of names. We focus on names because they are ubiquitous in familiar informal work with abstract syntax, so that it is natural to use them formalisations. Furthermore, Gabbay & Pitts have recently proposed a convincing set-theoretic approach of formalising abstract syntax that makes essential use of names [38]. This approach serves as the starting point for the design of our type theory.

In the rest of this chapter we review existing work for the formalisation of abstract syntax and give an overview of the work in this thesis. Because we focus on formal systems with names, much of this chapter is devoted to working formally with names. After a brief review of the use of names in informal work, we sketch the kinds of challenges one faces in the formalisation, and give an overview of existing approaches to working formally with names. This sets the scene for an overview of this thesis, given in Section 1.4. Finally, in the last section of this chapter we overview other approaches to the formalisation of abstract syntax that do not directly use names.

1.1 Names for Abstract Syntax

We take the syntax of the untyped λ-calculus as a running example. This is a good minimal example, since it is not far away from the syntax of real-world programming languages and since it gives rise to many of the problems that appear for real-world programming languages.

When defining the syntax of the λ-calculus, it is standard to first define a concrete version of the abstract syntax.

$$\mathsf{Lam} ::= \mathsf{var}\colon \mathbf{N} \mid \mathsf{app}\colon \mathsf{Lam} \times \mathsf{Lam} \mid \mathsf{lam}\colon \mathbf{N} \times \mathsf{Lam}$$

In this inductive definition, $\mathbf{N}$ is an appropriate type of variable names, such as the type of natural numbers or the type of character strings. As usual, we write x for $\mathsf{var}(x)$, $M\ N$ for $\mathsf{app}(M,N)$ and $\lambda x.M$ for $\mathsf{lam}(x,M)$.

Of course, the variable x in the term $\lambda x.M$ is meant to be bound. The terms are identified under (certain) renaming of bound variables. This means that the abstract syntax of the untyped λ-calculus is given by the quotient of Lam under α-congruence, where the α-congruence relation is the smallest congruence relation generated by $(\lambda x.M) \sim_\alpha (\lambda y.M[y/x])$ for $y \notin FV(M) \setminus \{x\}$.

In informal work, we usually make the fact that the syntax is given by a quotient under α-congruence as implicit as possible. Rather than working with equivalence classes, we work with particular representatives. This informal practice is captured by Barendregt's Variable Convention [9].

> 2.1.12. CONVENTION Terms that are α-congruent are identified. [...]
> 2.1.13. VARIABLE CONVENTION If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof) then in these terms all bound variables are chosen to be different from free variables. [...]
> 2.1.14. MORAL Using conventions 2.1.12 and 2.1.13 one can work with λ-terms in the naive way.

Barendregt's Variable Convention is very successful. It is pervasive in the literature and greatly simplifies informal work.

The formalisation of informal work that uses Barendregt's Variable Convention, however, turns out to be rather complicated. A large part of the formalisation is usually is spent on dealing with issues concerning names. Of course, this raises the question why it is that informal work is so much easier than formal work. Is it because informal work is imprecise and the details making formal work so much more complicated are swept under the carpet? Evidence that the answer to this question is (for the most part) no is given by the huge amount of existing informal work. There is little reason to doubt the correctness of this informal work, at least not for its use of Barendregt's Variable Convention. It therefore seems likely that informal reasoning is rigorous, but that the precise mathematical content of the informal arguments has not yet been fully identified. What is, for example, the *concept* of name binding? Clearly, there is more to binding than just the representation of α-equivalence classes. In informal work one also uses the operation which, given a name n and a term x, produces the α-equivalence class $n.x$. Another commonly used operation takes an α-equivalence class y and a sufficiently new name n and produces the instance $y@n$ of the class y at the name n. For another example, we should ask what the mathematical content of Barendregt's Variable Convention is. A direct formalisation would require us to *prove* that working with α-equivalence classes is really the same as working with instances chosen according to the convention.

In this thesis we study the concepts underlying informal arguments with names and design a dependent type theory based on this understanding.

1.2 What are the problems in formalising work with names?

To give an idea what kind of problems we encounter in the formalisation of informal work with names, let us look at a specific example. The example is the some/any equivalence identified by McKinna & Pollack [69] and Gabbay & Pitts [38]. It amounts to the observation that in informal work we often use the following reasoning. Suppose we have proved a property $\varphi(n)$ for some sufficiently fresh name n, and later we want to prove $\varphi(m)$ for some other fresh name m. We then usually say that in the proof of $\varphi(n)$ we could have used the name m instead of n, so that $\varphi(m)$ holds as well. This amounts to the

left to right direction of the equivalence below.

$$\exists\text{fresh name } n.\, \varphi(n) \iff \forall\text{fresh name } n.\, \varphi(n) \qquad \text{(some/any)}$$

Before giving a concrete example where such informal reasoning is used, let us first see why this reasoning should be correct. An informal 'proof' of the equivalence might go as follows.

'Proof' of (some/any).

($\Longrightarrow$) All we know about names is that there are infinitely many of them and that they can be compared. Given this, there is no way $\varphi(-)$ could possibly distinguish between two different fresh names. This is the case because a fresh name n is different from all the other names in $\varphi(n)$, so that any equality test with n in $\varphi(n)$ will always return false. Hence, if $\varphi(n)$ holds for some fresh name n, then so must $\varphi(m)$ for any other fresh name m.

($\Longleftarrow$) We can always find a fresh name.

□

Notice that in this 'proof' we did not have to look at the formula φ at all. Notice also that this 'proof' relies essentially on the fact that little can be done with names. If names are encoded as, say, natural numbers, then the informal proof breaks down, because in this case $\varphi(-)$ *could* distinguish two new names. This has an impact in practical applications. In McKinna & Pollack's development [69], for example, the information that little can be done with names is not available and the equivalence (some/any) has to be re-proven for each property φ.

A simple concrete example for the use of (some/any) is given by Vestergaard in a short note entitled 'Some/Any is Easy, Effective and Necessary' [115], see also [114]. The example is deciding whether or not two λ-terms in concrete syntax (not identified up to renaming of bound variables) are α-congruent. To illustrate the use of (some/any), it suffices to consider just the very special case of terms of the form $\lambda x.M$ and $\lambda y.N$ where neither M nor N contains an occurrence of λ. To decide whether $\lambda x.M$ and $\lambda y.N$ are α-equivalent, it obviously suffices to pick a name z not occurring in $\lambda x.M$ or $\lambda y.N$ and to check the syntactic identity $\lambda z.M[z/x] = \lambda z.N[z/y]$. In short, the algorithm computes the truth of the predicate $\exists z.\, z \notin \mathit{Vars}((\lambda x.M),(\lambda y.N)) \wedge \lambda z.M[z/x] = \lambda z.N[z/y]$. Call this predicate $\varphi(\lambda x.M, \lambda y.N)$. In order to prove that this algorithm really computes α-equality, we must, among other things, prove that it is transitive. For this, we have to show $\varphi(\lambda x.M, \lambda z.R)$, given the assumptions

$$\lambda u.M[u/x] = \lambda u.N[u/y] \qquad \lambda v.N[v/x] = \lambda v.R[v/z]$$

for some $u \notin \mathit{Vars}((\lambda x.M),(\lambda y.N))$ and some $v \notin \mathit{Vars}((\lambda y.N),(\lambda z.R))$. It is not obvious how to show $\varphi(\lambda x.M, \lambda z.R)$ directly from these assumptions, for example because u and z could be the same variable. Using the some/any property, on the other hand, we know that $\varphi(\lambda x.M, \lambda y.N)$ is equivalent

to $\forall z.\, z \notin \mathit{Vars}((\lambda x.M),(\lambda y.N)) \wedge \lambda z.M[z/x] = \lambda z.N[z/y]$. Write $\psi(\lambda x.M, \lambda y.N)$ for this universally quantified formula. Using the some/any equivalence, we can therefore make the assumptions

$$\lambda u.M[u/x] = \lambda u.N[u/y] \qquad \lambda u.N[u/x] = \lambda u.R[u/z]$$

for some $u \notin \mathit{Vars}((\lambda x.M),(\lambda y.N),(\lambda z.R))$, thus making transitivity trivial.

Note that there also would have been a problem with showing transitivity if we had used ψ instead of φ in the first place. In order to prove $\psi(\lambda x.M, \lambda z.R)$ from $\psi(\lambda x.M, \lambda y.N)$ and $\psi(\lambda y.N, \lambda z.R)$, we have to show $\lambda u.M[u/x] = \lambda u.R[u/z]$ for all names $u \notin \mathit{Vars}((\lambda x.M),(\lambda z.R))$, and there is a problem with using the assumption in case $u = y$ holds.

Other examples of some/any equivalences can be found in [69]. It is important to note that, while the above example concerns α-equivalence of terms, some/any equivalences are not only useful for α-equivalence classes of terms. Even when terms are identified up to α-equivalence, some/any reasoning appears, for example in the proof of the weakening property of pure type systems, see [69] and also the discussion in [86]. In general, some/any equivalences occur whenever we have to deal with object-level eigenvariables.

Formalising some/any equivalences is not the only challenge in the formalisation of informal work with names. The representation of names themselves is another one. The example of the some/any equivalence shows that it is essential for names to have very little structure. Because of this, it is not at all clear how to represent them. Taking names to be natural numbers, as is often done, is not ideal. Natural numbers have more internal structure than names, so that (some/any) does not always hold. More generally, it does not appear possible to define the type of names as an inductive type. It seems that the elements of any inductively defined type with infinitely elements will have more internal structure than just their identity.

Another challenge with the formalisation of informal work with abstract syntax is the support of inductive reasoning and recursive definitions. In informal work we commonly reason by structural induction on terms, and functions like substitution are (claimed to be) defined by structural recursion. If the abstract syntax does not contain binders, then these proofs and definitions can be formalised without problem. With variable binders, however, the syntax is defined as a quotient of an inductive type, and does not at first appear to be an inductive type. That it is really an inductive type has been shown in different settings in [30, 38, 49].

Finally, informal work makes extensive use of Barendregt's Variable Convention. To support it in formal work, we must identify terms up to α-equivalence, and we must (to some extent at least) be able to work with the α-equivalence classes as if they were concrete terms.

The above mentioned problems are some of the most important and well-known problems in the formalisation of names. There are likely to be more complications. For overviews of the problem and criteria formalisations of names have to meet see [89] and [6].

1.3 Approaches to the Formalisation of Names

Having outlined some of the challenges with the formalisation of names, we review representative examples of existing formalisations of abstract syntax with named variables. In this section we consider only approaches that use names; we refer to Section 1.5 for a discussion of other approaches.

1.3.1 Concrete Approaches

The first group of approaches is distinguished by the fact that the syntax is not identified up to renaming of bound variables. Two well-known representatives of this group are the approaches of McKinna & Pollack [69] and of Vestergaard & Brotherston [116]. In these approaches, names are represented by natural numbers and an abstraction $\lambda n.x$ is represented concretely, typically as a pair $\langle n, x\rangle$. Since the syntax is not identified under renaming of bound variables, α-conversion has to be taken care of explicitly. Moreover, the induction principles of the concrete syntax are not quite what would be expected in informal use. For example, neither McKinna & Pollack nor Vestergaard & Brotherston define capture-avoiding substitution as a function in the way this is usually done in informal work. Instead, substitution is possibly name-capturing but it is used only when it is safe to do so.

However, the fact that syntax is not identified up to α-conversion does not appear to be the main limiting factor in these approaches. Rather, the use of natural numbers for names appears to cause problems. McKinna & Pollack [69] show by using some/any equivalences that their approach can formalise informal arguments very well. However, because they encode names as numbers they must laboriously prove correct each instance of the some/any equivalences they use. As sketched above, in informal reasoning with names we expect such equivalences to come for free. In fact, McKinna & Pollack do not use any of the structure that numbers have over names, but they cannot exploit the fact that names have little structure, as would be necessary in order to get the some/any equivalences for free. Nevertheless, even with the problem of proving some/any equivalences, which may in the future be overcome by means of proof automation, McKinna & Pollack's approach is one of the best currently available for programming and reasoning with abstract syntax.

A big advantage of the concrete approaches is that everything we want to do with abstract syntax can indeed be done. This is not the case for all other approaches, an example being Higher Order Abstract Syntax described below.

There are proposals, similar in spirit to McKinna & Pollack's work, which address some of the issues of that work. For instance, to address the problem that terms are not identified up to renaming of bound variables, it is possible to work with an encoding of syntax that uses names for free variables but de Bruijn indices for bound variables [67]. This is essentially the approach of Gordon & Melham [41], being used, for example, by Norrish [78]. Norrish [79] also shows that nice recursive definitions are available for this encoding.

1.3.2 FM Theory

Next we discuss the nominal approach of Gabbay & Pitts [38]. Characteristic of this approach is its treatment of names. In the discussion of some/any equivalences we noted that it is important to model names well. Gabbay & Pitts address this issue. In order to model names well, they move from standard set theory to Fraenkel Mostowski (FM) set theory. In FM set theory there exists an infinite set whose elements can be compared for equality, but nothing else can be done with the elements. This set models names better than, say, natural numbers in standard set theory. For instance, Gabbay & Pitts show that some/any equivalences come for free when we use this set to model names.

Gabbay & Pitts' technical development rests on name-swapping actions. By requiring constructions to be invariant under name-swapping, they make available the fact that all we can do with names is to compare them. To develop their ideas, Gabbay & Pitts utilise FM set theory, an existing theory of sets with name-swapping actions. Gabbay & Pitts give a number of constructions that are useful in the formalisation of informal arguments with names.

- There is a syntax-independent notion of freshness, that is a binary relation $(-)\#(-)$ which formalises when two sets do not share names. It is syntax-independent because it is formulated using name-swapping, without any reference to abstract syntax.

- There is a freshness quantifier N that captures some/any equivalences. The formula $\mathit{N}n.\,\varphi(n)$ expresses both 'for *some* fresh name n the property $\varphi(n)$ holds' and 'for *all* fresh names n the property $\varphi(n)$ holds.' The quantifier N thus gives for free the equivalence (some/any) for arbitrary propositions φ.

- Abstraction sets $[\mathbf{N}]B$ give a representation of α-equivalence classes. Using abstraction sets abstract syntax up to α-conversion can be defined as an inductive type. Just as freshness is syntax-independent, abstraction sets can be used not only for terms but for arbitrary sets. A useful operation with abstraction sets is the binding operation taking a name $n \in \mathbf{N}$ and an element $x \in B$ to the α-equivalence class $n.x$.

As this brief list of construction shows, FM set theory addresses the issues with the formalisation of names that we have mentioned in the previous sections. We therefore believe that this approach is very promising—the development in this thesis is based on it.

The FM approach has the disadvantage that it requires us to move from a familiar universe, such as set theory, to a relatively unfamiliar one, such as FM set theory. As a consequence, it is not readily supported by existing theorem provers and programming languages. However, work is underway to build logics and programming languages based on the ideas developed in FM set theory. This thesis, in which we study how FM concepts can be built into dependent type theory, can be seen as part of this line of work. The set-theoretic setting in which Gabbay & Pitts develop their ideas is, however, not an ideal basis for this line of work. While a set-theoretic development may be advantageous for

accessibility, it does not lend itself to generalisations to other settings. We address this issue in this thesis by reformulating some of Gabbay & Pitts constructions categorically.

In the rest of this section, we give an overview of existing FM-related work.

Nominal Logic. Nominal Logic is a first order axiomatisation of the FM concepts that have been found useful for working with names. These concepts are name-swapping, freshness, the freshness quantifier and abstraction sets. Originally presented by Pitts [86] in a Hilbert-style axiomatisation, Gabbay and Cheney have also given presentations of Nominal Logic in sequent and natural deduction style [35, 37, 22]. While in Pitts' Hilbert-style presentation the main emphasis is on swapping, in the sequent and natural deduction presentations the emphasis is shifted towards freshness and the freshness quantifier. It is interesting to note that while the Hilbert-style presentation goes through very smoothly, the formulation of sequent and natural deductions rules for the freshness quantifier has presented some problems, as described by Cheney in [22].

By introducing Nominal Logic, Pitts shows that reasoning with names can be formalised in first-order logic. Restricting attention to first-order has the advantage over higher-order approaches, outlined at the end of this chapter, that problems are potentially easier to solve. One example for this is unification. In higher-order approaches to abstract syntax one often has to deal with higher-order unification, which is known to be much harder than first-order unification.

Nominal Unification and α-Prolog. Urban, Pitts & Gabbay [112] show that unification up to α-equivalence can be solved by a first-order unification algorithm. This unification algorithm is used by Cheney and Urban [23, 20] as the basis of a version of Prolog in which one can work with syntax up to α-equivalence. This programming languages implements logic programming with a version of Nominal Logic.

FM-HOL. Just as Nominal Logic axiomatises the structure of FM set theory in first-order logic, FM-HOL [34] axiomatises the structure of (a generalisation of) FM set theory in higher-order logic. Being higher order, it suffices to take names and name-swapping as primitives and define everything else from them, just as is done in FM set theory [38]. Although FM-HOL is not yet implemented, it can be hoped that it will make possible FM-style reasoning in theorem provers such as Isabelle/HOL.

FreshML 2000 and Fresh O'Caml. The concepts from FM set theory are also being studied in the context of functional programming. Pitts and Gabbay [87] have developed a ML-like programming language, FreshML 2000. In this language name-generation is treated in a side-effect free manner. This means that a new name can only be generated if the compiler can statically guarantee that the particular choice of name has no side-effect. The compiler guarantees the absence of side-effects by means of a type system containing freshness assertions. Freshness assertions are invisible to the user and are used only by the compiler. Practical experiments have shown that relying on the compiler to infer freshness properties automatically is often quite restrictive [100]. Since freshness assertions are invisible to the user, there is moreover no possibility for the user to help the compiler by supplying additional type annotations.

Shinwell, Pitts and Gabbay [102, 100] found that it is not necessary to have the compiler show absence of side-effects for name-generation. In [102, 101] it is shown that programming with syntax up to α-equivalence can be realised when name-generation is implemented as a side-effect. Using side-effects for name-generation appears sensible, since O'Caml also uses side-effects for other purposes such as I/O and memory access. The resulting full-scale programming language Fresh O'Caml [100] thus overcomes the problems of FreshML 2000.

Nominal Reasoning in Isabelle/HOL. All the FM-based systems described so far have the common aim of making it possible to work *inside* FM set theory or a similar universe in which Gabbay & Pitts' ideas can be developed. The development of Urban and Tasson [113] in Isabelle/HOL instead looks at such a universe from the outside. This work may be seen as the construction of a FM-like universe inside the higher-order logic of Isabelle/HOL, much in the way FM-Sets is constructed from Sets. This approach has the advantage that one is not constrained to working inside the FM-like universe. Indeed, in informal work with FM, such as [2, 101], such an external view on FM is often used. On the other hand, it remains to be seen how manageable the constructions of [113] in Isabelle/HOL are in practice. Initial experiments are encouraging.

Spatial Logic, Manipulating Trees with Hidden Labels. Names and FM concepts, such as the freshness quantifier, have found applications in a number of other areas. They have been studied, for example, in the context of spatial logics for concurrency theory [15, 17]. Here, we want to single out Cardelli, Gardner & Ghelli's work on the manipulation of trees with hidden labels [16], which has influenced the work in this thesis. We comment on the influence of this work in the description of our own approach below.

1.3.3 Weak Higher Order Abstract Syntax

Weak Higher Order Abstract Syntax (WHOAS) is another popular approach to formalising syntax with names in which terms are identified up to α-equivalence. The idea of WHOAS is to represent α-equivalence classes as functions from names to terms [29, 51]. Using this idea, the syntax of the untyped λ-calculus can be represented by the inductive type

$$\mathsf{Lam} ::= \mathsf{var}\colon \mathbf{N} \mid \mathsf{app}\colon \mathsf{Lam} \times \mathsf{Lam} \mid \mathsf{lam}\colon (\mathbf{N} \to \mathsf{Lam}).$$

With this encoding too, the problem arises of how the type of names $\mathbf{N}$ should be defined. Using natural numbers for $\mathbf{N}$ is again problematic. If we use numbers then the function space $(\mathbf{N} \to \mathsf{Lam})$ contains more than just α-equivalence classes. If numbers are used for names then Lam contains exotic terms that do not represent λ-terms, for example $\mathsf{lam}(\lambda n.\,\text{if } n = 3 \text{ then } \mathsf{var}(4) \text{ else } \mathsf{var}(5))$. One possibility of dealing with this problem is to use validity predicates to single out the functions in $(\mathbf{N} \to \mathsf{Lam})$ that do in fact represent α-equivalence classes. This approach is taken in [29]. It comes at the price that the abstract syntax corresponds only to a part of the inductive type Lam and that one has to deal with the validity predicates.

Another way of dealing with the problem that $(\mathbf{N} \to \mathsf{Lam})$ contains too many elements is to use a more restricted definition of $\mathbf{N}$ than the natural numbers. The above example of an exotic term shows that $\mathbf{N}$ cannot even have decidable equality. In fact, if $(\mathbf{N} \to \mathsf{Lam})$ is to contain only α-equivalence classes then there must not be anything we can do with names when defining a function of this type. Restricting the type of names is the approach taken by Fiore, Plotkin & Turi [30], Hofmann [49] and Honsell, Miculan & Scagnetto [51], among others. We discuss the approaches of Hofmann and Fiore et al., which are semantic in nature, in Section 1.5.3. Honsell et al. [51] introduce the Theory of Contexts [51], a higher-order logic in which nothing at all can be done with names for the definition of functions in $(\mathbf{N} \to \mathsf{Lam})$. However, if names were left completely unspecified then the resulting logic would not be expressive enough for the formalisation of informal arguments. The solution proposed in the Theory of Contexts is to add logical axioms for names, such as that any two names are either equal or different. The result is a logic that can formalise logical reasoning in a way similar to Nominal Logic but that is functionally weak in order to keep $(\mathbf{N} \to \mathsf{Lam})$ small. Hofmann [49] relates the Theory of Contexts to Nominal Logic by showing that both logics can be interpreted in the same semantic model. Although the Theory of Contexts has several disadvantages when compared to Nominal Logic, for example that the formulation of freshness is not syntax-independent and that the axiom of unique choice cannot be added, it has the important advantage that it can be used in existing theorem provers such as Coq.

1.4 Names and Binding in Type Theory

Missing from the above list of formal systems with names is a dependent type theory. The topic of this thesis is the design of such a dependent type theory with names.

A main reason for wanting such an integration is that dependent type theory is one of the most successful approaches to formal proof. It is used widely in theorem provers such as Coq, LEGO, Agda and NuPRL. Dependent type theory accommodates both functional programming and logical reasoning. Such an integration of programming and reasoning is desirable for the formalisation of informal work with names. When reasoning informally about abstract syntax, we often make use of functions like capture-avoiding substitution or the function computing the free variables of a term. For the purposes of this thesis we concentrate on first-order dependent type theory in the style of Martin-Löf [65, 77]. Under the Curry-Howard isomorphism, first-order dependent type theory corresponds to intuitionistic first-order logic. It has sum types $\Sigma x\colon A.\,B$ and product types $\Pi x\colon A.\,B$, which correspond to the propositions $\exists x\colon A.\,B$ and $\forall x\colon A.\,B$ of first-order logic.

To give a dependent type theory with names, we build on FM theory, which we believe best captures the concepts underlying informal work with names. Our aim is to give a type theory that contains useful FM constructions, such as abstraction sets and the freshness quantifier. The resulting type theory should make it possible to extend the Curry-Howard isomorphism from first-order logic to (at least part of) Nominal Logic. We approach the design of such a dependent type theory semantically, taking FM set

theory as our starting point.

Giving a type theory with names and FM constructions does, however, not appear to be simply a matter of replaying the definitions of Nominal Logic or FM-HOL. In type theory, we have to deal with a new concept, freefrom types, that does not appear in Nominal Logic, since it coincides with weakening there. Furthermore, existing FM-style logics are based on classical logic with extensional equality. Dependent type theory, on the other hand, usually corresponds to intuitionistic logic and extensional equality is known to be problematic [47]. Of the FM-based programming languages, Fresh O'Caml also does not appear to be a good model for a dependent type theory, since name-generation is modelled as a side-effect. The only remaining FM approach is FreshML 2000, and the dependent type theory presented in this thesis has some similarity to the type system of FreshML 2000.

Moreover, for dependent type theory it does not appear to be a good choice to take name-swapping as the only primitive for names, as is done in FM set theory. For example, to define a type of α-equivalence classes from swapping alone, it appears to be necessary to have extensional equality and some sort of quotient type. Because extensional equality causes problems with dependent types, we do not want to assume it just for the definition of α-equivalence. The alternative that we follow in this thesis is to treat constructions such as α-equivalence classes directly, turning them into primitives of the type theory. In addition to avoiding the above mentioned problems, a direct treatment of the FM constructs for names also contributes to a better understanding of the essential properties of these constructs.

For a direct formulation of FM concepts for names, we reformulate them in terms of freshness rather than swapping. This is not to be understood as an argument against swapping as such. The use of swapping by Gabbay, Pitts and others, e.g. in [38, 86, 16], is convincing. We have simply found freshness to be more convenient for the direct characterisation of FM concepts by universal properties.

1.4.1 Overview of our Approach

To build FM concepts into type theory, we begin by studying the categorical structure of the Schanuel topos, the categorical version of FM set theory. This is a first step towards giving a type theory, since there is a close connection between type theory and category theory, see e.g. [56].

We sketch the relevant categorical structure of the Schanuel topos and explain informally what it means. The central concept of our constructions is the freshness relation $(-)\#(-)$ as defined in FM set theory. Freshness gives rise to a monoidal structure $*$ by the definition

$$A * B \stackrel{\text{def}}{=} \{\langle x, y\rangle : A \times B \mid x\#y\}.$$

Taking this monoidal structure as our starting point, we define all other concepts for working with names by universal constructions in terms of it. In effect, there are only two constructions: non-standard sums Σ^* and dual non-standard products Π^*. Constructs such as abstraction sets, the freshness quantifier and unique choice of new names are all formulated in terms of Σ^* and Π^*.

The non-standard sums and products Σ^* and Π^* admit the following informal description. Recall that

the dependent product type $\Pi x\colon A.B$ is a type of functions that map each argument $M\colon A$ to an element of type $B[M/x]$. The type $\Pi^* x\colon A.B$ is a non-standard product type. It consists of partial functions mapping each argument $M\colon A$ that contains just fresh names to an element of $B[M/x]$. On arguments that do not contain just fresh names, the partial functions in $\Pi^* x\colon A.B$ are undefined. Being a sort of function type, there are versions of functional abstraction and application for Π^*, denoted by $\lambda^* x\colon A.M$ and $N@R$. Now recall that the dependent sum type $\Sigma x\colon A.B$ is a type of pairs $\langle M,N\rangle$ where $M\colon A$ and $N\colon B[M/x]$. The type $\Sigma^* x\colon A.B$ is a non-standard sum type. It consists of pairs $M.N$ in which all the names occurring in M have been hidden. This name-hiding can also be understood as name-binding: all the names in M are bound in the pair $M.N$. The type $\Sigma^* x\colon A.B$ is a dependent version of the abstraction set introduced by Gabbay & Pitts. The operations for working with Σ^*-types are similar to that for weak Σ-types [56, §10]. There are (in effect) a pair formation operation $M.N$ and an elimination operation (let M be $x.y$ in N). The computational intuition behind these operations is that $M.N$ first forms the pair $\langle M,N\rangle$ and then binds/hides the names of M in the pair, while (let M be $x.y$ in N) takes the pair-with-hidden-names M of type $\Sigma^* x\colon A.B$, replaces the bound/hidden names with new names and then binds the components of the pair to the variables x and y in N.

The main use of Σ^* and Π^* is to encode α-equivalence classes. As in FM set theory, we have a type $\mathbf{N}$ of names, the elements of which can be tested for equality, but nothing else can be done with them. The type $\Pi^* n\colon \mathbf{N}.\mathsf{Lam}$ represents α-equivalence classes of terms, much in the spirit of WHOAS. Notice that the standard function space $\Pi n\colon \mathbf{N}.\mathsf{Lam}$ contains more than just α-equivalence classes since names can be compared for equality. The non-standard sum type $\Sigma^* n\colon \mathbf{N}.\mathsf{Lam}$ consists of pairs $n.t$ of a name n and a term t such that n is bound in the pair. In this way, the type $\Sigma^* n\colon \mathbf{N}.\mathsf{Lam}$ also contains α-equivalence classes of terms. It is nothing but the FM abstraction set. With Π^* being the dual of Σ^*, we can therefore say that WHOAS-style syntax is dual to FM-style syntax.

Categorically, Σ^* and Π^* may be understood as monoidal versions of Σ and Π. Standard sums and products Σ and Π are defined as left and right adjoints to weakening $\Sigma_A \dashv \pi_A^* \dashv \Pi_A$, subject to a Beck-Chevalley condition. The non-standard versions Σ^* and Π^* arise if in this situation we replace the weakening functor π_A^* by a non-standard 'weakening' functor W_A. This functor W_A is a lifting of the monoidal structure $*$. In Chapter 2 we review the categorical structure corresponding to dependent types and define Σ^*, W and Π^* precisely. Following this, in Chapter 3 we construct concrete models of this structure, namely the Schanuel topos and a realizability version of the Schanuel topos.

Based on this categorical analysis, we give a dependent type theory containing the normal dependent sums Σ and products Π as well as the non-standard monoidal sums Σ^* and products Π^*. This is the topic of Chapters 4–9. As in the semantics, the basis of our development is the monoidal structure $*$. We build it into the type theory using bunched contexts [80, 91]. The definition of the basic type theory appears in Chapters 4 and 5. In addition to the usual additive types, this type theory contains Π^*-types and $*$-types. In Chapters 6 and 7 we then show it sound and complete for a categorical semantics. Our use of bunches and Π^*-types was inspired by the work of Cardelli, Gardner & Ghelli [16].

Having introduced Π^*-types, the introduction of Σ^*-types is the topic of Chapters 8 and 9. First, in Chapter 8 we introduce syntax for the functor W, as appears to be necessary for giving a syntax for Σ^*. The syntax for W takes the form of *freefrom* types. The freefrom type $B^{*(M:A)}$ intuitively consists of all elements of B that are free from the names in the value denoted by $M\colon A$. In the type system of FreshML 2000 a similar use of freshness appears. The main difference to our approach is that with type dependency freefrom types can be made first-class types, whereas in FreshML 2000 the freshness assertions are visible only to the compiler. In FreshML 2000 the user has to rely on the compiler to infer freshness automatically—and the compiler is not always able to do this. With first-class freefrom types, on the other hand, the user has the option of supplying freshness properties explicitly.

The monoidal types Π^* and Σ^* are the basis of our approach to working with names and binding in type theory. In Chapter 10, we explain categorically how the concept of name-binding can be formalised in terms of them. In order to do so, we give a categorical formulation of Barendregt's Variable Convention. The central idea of this formulation is a categorical equivalence that formalises Barendregt's requirement that working with α-equivalence classes is essentially the same as working with freshly named concrete terms. This equivalence amounts to an isomorphism $\Sigma^* n\colon \mathbf{N}.B \cong \Pi^* n\colon \mathbf{N}.B$, where $\mathbf{N}$ is the type of names. This isomorphism can be understood as a some/any equivalence. Under the propositions-as-types reading, the type $\Sigma^* n\colon \mathbf{N}.B$ expresses 'B holds for some new name n' and the type $\Pi^* n\colon \mathbf{N}.B$ expresses 'B holds for all new names n.' The isomorphism then states that these two propositions are equivalent, thus amounting to a some/any equivalence. This situation generalises the freshness quantifier $\mathcal{N}$ of Nominal Logic. If we restrict to subobjects then $\Sigma^* n\colon \mathbf{N}.\varphi$ and $\Pi^* n\colon \mathbf{N}.\varphi$ both specialise to $\mathcal{N}n.\varphi$. Another way of understanding the isomorphism is by means of the fact that both $\Sigma^* n\colon \mathbf{N}.B$ and $\Pi^* n\colon \mathbf{N}.B$ represent α-equivalence classes of elements of B with respect to a single name n. If both represent α-equivalence classes, then there should of course be an isomorphism between them. We show in Chapter 10 that this categorical structure includes a good deal of the structure used for programming and reasoning with names in FM set theory. Our structure may be seen as a propositions-as-types generalisation of Nominal Logic.

Having identified the conceptual structure of names and name-binding, in Chapter 11 we extend the type theory with it. We incorporate the isomorphism $\Sigma^* n\colon \mathbf{N}.B \cong \Pi^* n\colon \mathbf{N}.B$ in the type theory by means of *hidden-name types* $\mathrm{H}n.B$. The type $\mathrm{H}n.B$ may be seen as being both $\Sigma^* n\colon \mathbf{N}.B$ and $\Pi^* n\colon \mathbf{N}.B$ at the same time[1]. Hidden-name types have the term constructors and destructors from both Σ^* and Π^*. There are isomorphisms $(\Sigma^* n\colon \mathbf{N}.B) \cong (\mathrm{H}n.B) \cong (\Pi^* n\colon \mathbf{N}.B)$. The type $\mathrm{H}n.B$ thus plays the same role as $\mathcal{N}n.\varphi$ in Nominal Logic.

Finally, in Chapter 12 we give some examples to illustrate how the type theory can be used for working with names and binding.

A short description of the above approach has been published in a joint paper with Ian Stark [95].

[1] The letter H, being half Π and half $\amalg$, is the ideal symbol for a type that is both a product and a sum. The name is also inspired by the hidden-name quantifier of Cardelli and Gordon [17]. Due to differences in setting, the precise relation of hidden-name types to the hidden-name quantifier is however not clear.

1.5 Other Approaches to the Formalisation of Abstract Syntax

As motivated above, in this thesis we focus on nominal, i.e. FM-based, approaches to the formalisation of abstract syntax. However, as previous work as well as the work reported in this thesis shows, the design of nominal logics and type systems is not without problem. We should therefore keep in mind other approaches to the formalisation of abstract syntax. In the rest of this section we review some of these non-nominal approaches.

1.5.1 De Bruijn Indices

One of the most commonly used ways of encoding α-equivalence classes of terms is with Bruijn indices [14]. In this approach, bound variables are represented by a natural number counting the distance to its binder.

$$\mathsf{Lam} ::= \mathsf{var}\colon \mathbb{N} \mid \mathsf{app}\colon \mathsf{Lam} \times \mathsf{Lam} \mid \mathsf{lam}\colon \mathsf{Lam}$$

Free variables can be treated by viewing them as being implicitly bound at the top level. It is also possible to treat them separately using variable names, as in e.g. [67].

The advantages of de Bruijn's approach are that it identifies terms up to α-equivalence, it is available in virtually all programming languages and logics, and it makes arbitrary operations with syntax possible. The most important disadvantage of de Bruijn indices is that the formal treatment of syntax does not correspond directly to the informal treatment. This makes the definition of basic operations such as substitution complicated and error-prone. This may be no more than a nuisance in programming, since such operations may have to be defined only once, perhaps even automatically, but it is a problem in formal reasoning where humans are expected to work with particular terms.

For these reasons, we believe that de Bruijn indices are not the ideal solution for the representation of abstract syntax. That said, syntax with de Bruijn indices can be very helpful as a technical tool. For example, internal languages of categories are most naturally formulated with de Bruijn indices, see e.g. Chapter 6.

1.5.2 Higher Order Abstract Syntax

The idea of Higher Order Abstract Syntax (HOAS), which goes back to Church [24], is to encode variable binders using function spaces.

$$\mathsf{Lam} ::= \mathsf{var}\colon \mathbf{N} \mid \mathsf{app}\colon \mathsf{Lam} \times \mathsf{Lam} \mid \mathsf{lam}\colon (\mathsf{Lam} \to \mathsf{Lam})$$

This encoding is extremely successful for the representation of formal systems in logical frameworks. It is used in Isabelle [82], the LF logical framework [42] and λProlog [75], to mention just a few. Such HOAS encodings of formal systems are very convenient, since properties such as closure under weakening and substitution come for free.

Higher Order Abstract Syntax is less successful if the aim is not just to represent the abstract syntax but also to reason about it. In order for an HOAS encoding to work, the function space $(\mathsf{Lam} \to \mathsf{Lam})$ should contain no more than α-equivalence classes of terms. This means that we cannot allow elements of $(\mathsf{Lam} \to \mathsf{Lam})$ to be defined by primitive recursion. Moreover, because Lam occurs both positively and negatively in $(\mathsf{Lam} \to \mathsf{Lam})$, the type Lam cannot be defined directly as an inductive type. Despeyroux et al. [29] have introduced WHOAS, as discussed in Section 1.3.3, as a way of side-stepping this problem and defining the syntax as an inductive type. A further important problem of using HOAS for working formally with syntax is that HOAS has no notion of object-level variable. For example, it is not clear how to the define the function that computes the free variables of a term. Informal arguments that explicitly mention variables must therefore be reformulated, and it is not clear if this is always possible. Finally, one of the advantages of HOAS for the representation of logical systems, that closure under substitution comes for free, can also become a disadvantage, namely when the logical system to be encoded does not have this property. For example, many properties of the π-calculus are not closed under arbitrary name-substitutions but only under injective ones. This problem has lead to the development of substructural logical frameworks, see e.g. [92, 19].

We describe some of the existing solutions to some of the issues with HOAS.

1.5.2.1 Multi-level Reasoning

The problem that adding induction principles to a logical framework gives rise to exotic HOAS terms can be avoided by moving to multi-level reasoning. The idea is to represent abstract syntax using HOAS in a logical framework without induction or recursion and to use a second, separate, logical framework to reason about the first one. In the second logical framework induction and recursion principles can be added without problem, as they cannot be used to define elements of $(\mathsf{Lam} \to \mathsf{Lam})$ in the first system.

Multi-level reasoning is proposed in the logics $FO\lambda^{\Delta\mathbb{N}}$ of McDowell & Miller [68] and $FO\lambda^{\Delta\nabla}$ of Miller & Tiu [73]. These closely related approaches use higher order abstract syntax in a simply-typed λ-calculus and support logical reasoning by means of a sequent calculus on top of the λ-calculus. Interestingly, $FO\lambda^{\Delta\nabla}$ also contains a quantifier ∇ that can be used to deal with eigenvariables. Using this quantifier Miller & Tiu [73] can formalise reasoning with the π-calculus without the problem of non-injective name-substitutions. Furthermore, induction and coinduction can be added to this setting [74]. A prototype implementation Linc is being developed by Tiu.

Multi-level reasoning is also available in Twelf [97], a logic programming language with experimental theorem proving facilities.

At present, the existing multi-level approaches are closer to logic programming than to theorem proving. None of the above approaches is intended for interactive proof. Furthermore, while these multi-level approaches look very promising for a wide range of applications, they do not apparently deal with the problem of HOAS that informal arguments making explicit reference to variables names cannot be formalised directly.

1.5.2.2 Modal Type Systems

In order to maintain adequacy of syntax encodings, HOAS theories like the above mentioned multi-level approaches or the Theory of Contexts have a weak functional theory. Therefore, instead of working with functions one can only work with functional relations.

A proposal to strengthen the functional theory without introducing exotic terms is the modal type system of Schürmann, Despeyroux & Pfenning [98]. The aim of this type system is to allow structural recursion, but to manage the definitions so that no exotic terms are introduced. This is achieved by means of a modality $\Box$ on types. The function space $(\mathsf{Lam} \to \mathsf{Lam})$ has the same elements as before, but $((\Box\mathsf{Lam}) \to \mathsf{Lam})$ also contains functions defined by structural recursion.

1.5.2.3 HOAS or Names?

Can we conclude by saying that HOAS or the nominal approaches are better than the other? At present, we do not think this question can be answered conclusively, certainly not for all possible situations. We do think the concepts underlying informal work with abstract syntax are best explained by nominal approaches. The development of Fresh O'Caml is an example of the power of nominal approaches. On the other hand, recent developments like the ∇-quantifier show that reasoning with HOAS can be both simple and powerful. Furthermore, there appears to be some convergence of HOAS and nominal approaches, as exemplified by the similar quantifiers ∇ and $И$. We suggest a possible common framework for understanding ∇ and $И$ in Chapter 10. Specifically, we believe that ∇ and $И$ are both instances of Proposition 10.1.4, where ∇ arises if we take (linear) species as the underlying category and $И$ arises if we use the Schanuel topos. The verdict on whether to prefer HOAS or nominal approaches is still out, although it seems doubtful that one approach will turn out to be superior in all respects.

1.5.3 Semantic Approaches

Since the starting point of this thesis is the semantics of names and binding, we should also mention other semantic approaches that have influenced the development.

In [49] Hofmann studies the semantics of HOAS, modelling induction principles, modal type systems for structural recursion as well as the Theory of Contexts. His work clarifies the concepts of these systems and their interrelation. He shows that these systems can be modelled in different presheaf categories. By modelling the logic of Theory of Contexts in the Schanuel topos, he also provides a formal link between WHOAS and FM theory.

Fiore, Plotkin & Turi [30] give an algebraic account of abstract syntax with variable binders, showing that well-known theory from universal algebra can be generalised to work in the presence of variable binders. Their solution is a version of WHOAS. The main focus of the work in [30] is on representing syntax, implementing substitution and describing initial algebra semantics. A similar development for linear binders is given by Tanaka [107].

Power and Tanaka [90, 108] have unified the approaches in [30] and [107] and have given a universal account of abstract syntax with variable binders and substitution.

Although the development in this thesis has greatly benefited from the understanding of these semantic approaches, there is a difference in purpose. In this thesis we consider the semantic concepts of names and name-binding, while the above mentioned approaches focus on the representation of syntax with binding and the implementation of substitution. We further discuss the relationship to these approaches in Section 10.4.

The concepts of names and name-binding, as they appear in FM theory, have been studied directly by Menni [72]. His work has directly influenced the work reported in this thesis, and is discussed in more detail in Chapter 10.

1.6 Synopsis

The content of this dissertation falls into two parts. In the first part, Chapters 2–9, we introduce and study a bunched dependent type theory with Π^*-types and Σ^*-types. We study this type theory in general, leaving open the monoidal structure $*$ from which Π^* and Σ^* are derived. In Chapter 2 we start with the categorical definition of Π^* and Σ^*; in Chapter 3 we construct particular models for this structure; in Chapters 4 and 5 we introduce a bunched dependent type theory with Π^*-types; in Chapters 6 and 7 we show soundness and completeness for this type theory; and in Chapters 8 and 9 we extend the type theory with freefrom-types and Σ^*-types.

The second part of this thesis, Chapters 10–12, is about names and name-binding in dependent type theory. We show how the type theory from the first part can be used for working with names and binding. In Chapter 10 we give a categorical description of names and name-binding in terms of Π^* and Σ^*; in Chapter 11 we use this categorical description to extend the type theory from the first part with names and binding; and in Chapter 12 we give examples of how the type theory can be used for working with abstract syntax.

Chapter 2

Fibrations, Dependent Types and Monoidal Structure

In the first part of this dissertation, up to Chapter 9, we study how a monoidal structure can be built into dependent type theory. We look at this problem both from a semantic and a syntactic perspective. In the present chapter we start by studying the categorical structure of dependent types with a monoidal structure; in Chapter 3 we construct specific instances of this structure; and in Chapters 4–9 we introduce and study a syntax for it.

In this chapter we look at the categorical structure underlying dependent types. After giving basic definitions for categories and fibrations, we consider how a monoidal structure fits into this picture. We define monoidal sums $\Sigma^{\otimes}$ and monoidal products $\Pi^{\otimes}$, which are the basis of the later development.

2.1 Preliminaries on Categories

We assume the reader to be familiar with the basic concepts of category theory [63, 111].

We make the following notational conventions.

- Categories are denoted by $\mathbb{B}, \mathbb{C}, \mathbb{D}, \mathbb{E}, \ldots$, objects by capital letters $A, B, C, \ldots$ and morphisms by lower-case letters $f, g, h, u, v, \ldots$. We write id_A or simply id for the identity on A, and write $g \circ f$ for the composition of $f\colon A \to B$ and $g\colon B \to C$. The collection of morphisms from A to B in $\mathbb{B}$ is denoted by $\mathbb{B}(A, B)$.
- **Sets** is the category of sets and functions.
- The arrow category $\mathbb{B}^{\to}$ has as objects the maps in $\mathbb{B}$. The morphisms in $\mathbb{B}^{\to}(f\colon A \to B, g\colon C \to D)$ are given by pairs $(u\colon A \to C, v\colon B \to D)$ satisfying $v \circ f = g \circ u$.

- The slice category $\mathbb{B}/A$, where A is an object of $\mathbb{B}$, is the subcategory of $\mathbb{B}^{\to}$ in which the objects are maps with codomain A and the morphisms have the form (u, id).
- We write $Sub(\mathbb{B})$ for the category of subobjects in $\mathbb{B}$. For an object A in $\mathbb{B}$, we write $Sub_{\mathbb{B}}(A)$ or just $Sub(A)$ for the poset of subobjects on A.
- The category of functors from $\mathbb{B}$ to $\mathbb{C}$ is denoted by $\mathbb{C}^{\mathbb{B}}$. We often write Id for the identity functor.
- Cartesian products are denoted by $A \times B$. We write $\pi_1 \colon A \times B \to A$ and $\pi_2 \colon A \times B \to B$ for the projections. Given maps $f \colon C \to A$ and $g \colon C \to B$, we write $\langle f, g \rangle \colon C \to A \times B$ for the unique map satisfying $\pi_1 \circ \langle f, g \rangle = f$ and $\pi_2 \circ \langle f, g \rangle = g$.
- Coproducts are denoted by $A + B$. We write $\kappa_1 \colon A \to A + B$ and $\kappa_2 \colon B \to A + B$ for the coprojections. We write $[f, g] \colon A + B \to C$ for the copairing of two morphisms $f \colon A \to C$ and $g \colon B \to C$.
- We use the notation $(A \Rightarrow B)$ for the cartesian closed structure, writing $\mathrm{ev} \colon (A \Rightarrow B) \times A \to B$ for the counit of $A \times (-) \dashv (A \Rightarrow -)$. Given $f \colon C \times A \to B$, we write $\Lambda f \colon C \to (A \Rightarrow B)$ for the abstraction of f.
- We write $F \dashv U$ if F is left adjoint to U. The unit and counit are denoted by $\eta \colon Id \to UF$ and $\varepsilon \colon FU \to Id$ respectively. Given $f \colon A \to UB$ and $g \colon FC \to D$, we write $f^{\sharp} \colon FA \to B$ and $g^{\flat} \colon C \to UD$ for the adjoint transposes.

We follow the terminology of Mac Lane [63].

Definition 2.1.1. A category $\mathbb{B}$ has *universal coproducts* if it has binary coproducts and these are preserved by pullback, that is if the two squares in the left diagram below are pullbacks then the top row is a coproduct diagram. Coproducts are *disjoint* if the coprojections are mono and the square on the right below is a pullback.

$$\begin{array}{ccccc} A & \longrightarrow & B & \longleftarrow & C \\ \downarrow & & \downarrow & & \downarrow \\ D & \xrightarrow[\kappa_1]{} & D+E & \xleftarrow[\kappa_2]{} & E \end{array} \qquad \begin{array}{ccc} 0 & \longrightarrow & B \\ \downarrow & & \downarrow{\scriptstyle \kappa_2} \\ A & \xrightarrow[\kappa_1]{} & A+B \end{array}$$

We use the following formulation of Beck's theorem.

Theorem 2.1.2 (Beck)**.** *Let $U : \mathbb{C} \to \mathbb{B}$ be a functor. Then U is comonadic if all of the following conditions are satisfied.*

1. *U has a right adjoint.*
2. *U is faithful and reflects isomorphisms.*
3. *$\mathbb{C}$ has equalisers and U preserves them.*

Proof. See e.g. [57, Theorem 1.1.2]. □

At some points (e.g. in Chapter 10) we will work with the subobject logic of categories. For our purposes, it suffices to use coherent logic, i.e. first-order logic with the connectives $\top$, $\bot$, $\wedge$, $\vee$ and $\exists$.

Definition 2.1.3. A *cover* is a map c with the property that, whenever $c = m \circ c'$ holds for a monomorphism m, then m is an isomorphism.

Definition 2.1.4. A category $\mathbb{B}$ with finite limits is *regular* if each map factors into a cover followed by a monomorphism, and covers are stable under pullback.

In a regular category the covers are just the regular epimorphisms, that is epimorphisms that occur as coequalisers.

Definition 2.1.5. A category $\mathbb{B}$ is *coherent* if it is regular and the following two conditions hold.

- For each object A, the subobject poset $Sub(A)$ has binary joins $\vee$ and these are preserved by pullbacks.
- $\mathbb{B}$ has an initial object 0 and each arrow $A \to 0$ is an isomorphism.

See e.g. [57, 56] for more information on coherent categories.

2.2 Preliminaries on Fibrations

In this section we fix the notation on fibrations. All the information on fibrations needed in this thesis can be found in Jacobs' book [56], from where the definitions in this section are taken. We (largely) follow Jacobs' notation and terminology.

2.2.1 Fibrations

Definition 2.2.1 (Fibration)**.** Let $p\colon \mathbb{E} \to \mathbb{B}$ be a functor.

1. A morphism $f\colon A \to B$ in $\mathbb{E}$ is *cartesian over* $u\colon \Gamma \to \Delta$ in $\mathbb{B}$ if $pf = u$ holds and for every morphism $g\colon C \to B$ and every morphism $v\colon pC \to \Gamma$ for which $pg = u \circ v$ holds, there exists a unique map $h\colon C \to A$ satisfying $g = f \circ h$ and $v = ph$.
2. The functor $p\colon \mathbb{E} \to \mathbb{B}$ is a *fibration* if for each object B in $\mathbb{E}$ and each morphism $u\colon \Gamma \to pB$ in $\mathbb{B}$ there exists a morphism $f\colon A \to B$ cartesian over u.

We say a map f in $\mathbb{E}$ is *vertical* if $pf = id$ holds. For any object Γ in $\mathbb{B}$ we write $\mathbb{E}_\Gamma$ for the *fibre over* Γ, that is the category of objects over Γ and vertical morphisms between them. For a fibration $p\colon \mathbb{E} \to \mathbb{B}$, the categories $\mathbb{B}$ and $\mathbb{E}$ are usually called *base category* and *total category* respectively.

Of particular importance for this thesis is the codomain fibration $cod\colon \mathbb{B}^{\to} \to \mathbb{B}$ for a category $\mathbb{B}$ having all pullbacks. In cod the cartesian maps are given by pullback squares. See [56] for more examples of fibrations.

Definition 2.2.2 (Cloven Fibration)**.** A fibration $p\colon \mathbb{E} \to \mathbb{B}$ is *cloven* if, for each object B in $\mathbb{B}$ and each map $u\colon \Gamma \to pB$ in $\mathbb{B}$, we have a chosen map $\overline{u}(B)\colon u^*(B) \to B$ over u. The choice of lifting is called *cleavage*.

The cleavage defines, for each map $u\colon \Gamma \to \Delta$ in $\mathbb{B}$, a reindexing functor $u^*\colon \mathbb{E}_\Delta \to \mathbb{E}_\Gamma$. Following from the fact that the maps $\overline{u}(B)$ are cartesian, there are canonical vertical natural isomorphisms $Id \xrightarrow{\cong} id^*$ and $v^* \circ u^* \xrightarrow{\cong} (u \circ v)^*$ for all $u\colon \Gamma \to \Delta$ and $v\colon \Delta \to \Phi$, see [56, §1.4].

Definition 2.2.3 (Split Fibration)**.** A cloven fibration is *split* if the canonical isomorphisms $Id \xrightarrow{\cong} id^*$ and $v^* \circ u^* \xrightarrow{\cong} (u \circ v)^*$ are the identity.

In a split fibration, the choice of cartesian liftings is called *splitting*.

2.2.2 Change of Base

Given a fibration $p\colon \mathbb{E} \to \mathbb{B}$ and a functor $K\colon \mathbb{C} \to \mathbb{B}$, we can form a fibration K^*p by change-of-base of p along K. The fibration K^*p is defined by the following pullback of categories.

$$\begin{array}{ccc} K^*\mathbb{E} & \xrightarrow{L} & \mathbb{E} \\ {\scriptstyle K^*p}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ \mathbb{C} & \xrightarrow[K]{} & \mathbb{B} \end{array}$$

If p is a split fibration then so is K^*p. Examples of this situation can be found in [56, §1.5].

In this thesis we are concerned mostly with the special change-of-base situation where p is a codomain fibration. This situation is known as gluing, see e.g. [110, §7.7]. It is given by the following diagram.

$$\begin{array}{ccc} \mathbb{B}/K & \xrightarrow{L} & \mathbb{B}^{\to} \\ {\scriptstyle Gl(K)}\downarrow & \lrcorner & \downarrow{\scriptstyle cod} \\ \mathbb{C} & \xrightarrow[K]{} & \mathbb{B} \end{array}$$

The category $\mathbb{B}/K$ is a comma category [63]. Its objects are triples $(A, B, f\colon A \to KB)$ in which A is an object of $\mathbb{B}$, B is an object of $\mathbb{C}$ and f is a morphism in $\mathbb{B}$. Its morphisms from $(A, B, f\colon A \to KB)$ to $(C, D, g\colon C \to KD)$ are pairs $(u\colon A \to C, v\colon D \to B)$ of morphisms in $\mathbb{B}$ and $\mathbb{C}$ for which $g \circ u = Kv \circ f$ holds. The functor $Gl(K)$ maps an object (A, B, f) to B and a morphism (u, v) to v. The functor L maps (A, B, f) to f and (u, v) to (u, Kv).

2.2.3 Categories of Fibrations

Definition 2.2.4 (Fibred Functor)**.** Let $p\colon \mathbb{E} \to \mathbb{B}$ and $q\colon \mathbb{D} \to \mathbb{B}$ be fibrations over $\mathbb{B}$. A *fibred functor* from p to q is a functor $F\colon \mathbb{E} \to \mathbb{D}$ that satisfies $q \circ F = p$ and that preserves cartesian maps. If both p

and q are split fibrations and F preserves the splitting up to equality, i.e. $F(\overline{u}(B)) = \overline{u(F(B))}$ holds for all $u\colon \Gamma \to \Delta$ in $\mathbb{B}$ and all B in $\mathbb{E}_\Delta$, then F is called a *split fibred functor*.

Definition 2.2.5 (Fibred Natural Transformation)**.** A *fibred natural transformation* $\tau\colon F \to G$ between fibred functors $F, G\colon p \to q$ is a natural transformation from F to G whose components are all vertical. We sometimes speak of a *vertical natural transformation*.

These definitions give rise to a 2-category $Fib(\mathbb{B})$ of fibrations on base $\mathbb{B}$. The objects (0-cells) are fibrations, the morphisms (1-cells) are fibred functors and the 2-cells are fibred natural transformations. If all this structure is required to be split, then this defines a 2-category $Fib_{split}(\mathbb{B})$. More information on the structure of these categories of fibrations can be found in [45, 56, 44, 54].

2.2.4 Fibred Adjunctions

Definition 2.2.6 (Fibred Adjunction)**.** Let $p\colon \mathbb{E} \to \mathbb{B}$ and $q\colon \mathbb{D} \to \mathbb{B}$ be fibrations on $\mathbb{B}$. A *fibred adjunction* $F \dashv U$ is given by fibred functors $U\colon p \to q$ and $F\colon q \to p$ together with vertical natural transformations $\eta\colon Id \to UF$ and $\varepsilon\colon FU \to Id$ satisfying the triangular identities $U\varepsilon \circ \eta_U = id$ and $\varepsilon_F \circ F\eta = id$. If in this situation p and q are split fibrations and F and U are split fibred functors then we speak of a *split fibred adjunction*.

An example is the fibred terminal object functor $1\colon \mathbb{B} \to \mathbb{E}$ for a fibration $p\colon \mathbb{E} \to \mathbb{B}$. A fibration has fibred terminal objects if each fibre has terminal objects and these are preserved by reindexing. This is equivalent to saying that the unique fibred functor from p to the identity fibration $id_{\mathbb{B}}\colon \mathbb{B} \to \mathbb{B}$, which is given by the functor $p\colon \mathbb{E} \to \mathbb{B}$, has a fibred right adjoint $1\colon p \to id_{\mathbb{B}}$. This terminal object functor 1 is just the functor $1\colon \mathbb{B} \to \mathbb{E}$ mapping Γ to the terminal object of $\mathbb{E}_\Gamma$.

Lemma 2.2.7. *Let $p\colon \mathbb{E} \to \mathbb{B}$ and $q\colon \mathbb{D} \to \mathbb{B}$ be fibrations over $\mathbb{B}$. A fibred functor $U\colon p \to q$ has a fibred left adjoint if and only if the following two conditions hold.*

1. *For each Γ, the restriction $U_\Gamma\colon \mathbb{E}_\Gamma \to \mathbb{D}_\Gamma$ of U to the fibre over Γ has a left adjoint F_Γ.*
2. *For each morphism $u\colon \Gamma \to \Delta$ in $\mathbb{B}$, the canonical transformation $F_\Gamma u^* \to u^* F_\Delta$ is an isomorphism. This condition is known as a* Beck-Chevalley condition.

If p and q are split fibrations and U is a split fibred functor then U has a split fibred left adjoint if and only if the above two conditions hold and the isomorphism in 2 is the identity.

See [56, Lemma 1.8.9] for a proof.

The 2-category $Fib(\mathbb{B})$ provides us with the following notion of equivalence of fibrations.

Definition 2.2.8 (Equivalence of Fibrations)**.** Two fibrations $p\colon \mathbb{E} \to \mathbb{B}$ and $q\colon \mathbb{D} \to \mathbb{B}$ are *equivalent* if there are fibred functors $U\colon p \to q$ and $F\colon q \to p$ and vertical natural isomorphisms $Id \cong UF$ and $FU \cong Id$.

We note that the notion of equivalence obtained from $Fib_{split}(\mathbb{B})$ differs from the above insofar that not only the fibrations p and q must be split but also the functors U and F.

Lemma 2.2.9. *Two fibrations $p\colon \mathbb{E} \to \mathbb{B}$ and $q\colon \mathbb{D} \to \mathbb{B}$ are equivalent if and only if there exists a fibred functor $U\colon p \to q$ which, as a functor from $\mathbb{E}$ to $\mathbb{D}$, is full, faithful and essentially surjective on objects.*

Proof. It p and q are equivalent then the existence of a suitable functor U is immediate.

In the other direction, we have to show that any functor U with the stated properties gives rise to an equivalence of p and q. Consider the restriction $U_\Gamma\colon \mathbb{E}_\Gamma \to \mathbb{D}_\Gamma$ of U to the fibre over Γ. It is immediate that U_Γ is full and faithful since U is. Now we show that U_Γ is essentially surjective on objects. Let B be an object in $\mathbb{D}_\Gamma$. Since U is essentially surjective, there exists an object A in $\mathbb{E}$ and a (not necessarily vertical) isomorphism $i\colon B \to UA$ in $\mathbb{D}$. Then, the map qi in $\mathbb{B}$ is also an isomorphism, and, since cartesian maps over isomorphisms are isomorphisms, so is $\overline{(qi)}\colon (qi)^*UA \to UA$ in $\mathbb{D}$. Hence, by the universal property of cartesian morphisms, there exists an isomorphism $j\colon B \to (qi)^*UA$ in $\mathbb{D}_\Gamma$. Since U is a fibred functor, there exists an isomorphism $(qi)^*UA \to U(qi)^*A$ in $\mathbb{D}_\Gamma$. Noting that $(qi)^*A$ is an object of $\mathbb{E}_\Gamma$, we have thus shown that B is isomorphic (in $\mathbb{D}_\Gamma$) to an object in the image of U_Γ, i.e. that U_Γ is essentially surjective.

Since, for each Γ, U_Γ is full, faithful and essentially surjective, we obtain that U_Γ has a left adjoint F_Γ and the (vertical) unit and counit of this adjunction are isomorphisms, see e.g. [63, §Theorem IV.4.1]. The result now follows from Lemma 2.2.7. The Beck-Chevalley condition holds because the canonical map $F_\Gamma u^* \to u^* F_\Delta$ is, by definition, such that it is an isomorphism if both the unit and counit of $F_\Gamma \dashv U_\Gamma$ are. □

Note that this argument does not work for a split equivalence, since even if we assume p, q and U to be split, we can only assume the map $F_\Gamma u^* \to u^* F_\Delta$ to be an isomorphism and not the identity.

2.3 Comprehension Categories

In this section we give the definition of comprehension categories, proposed by Jacobs as a general framework for modelling type dependency, see [54, 55] and [56, §10.4]. Comprehension categories generalise the structure of locally cartesian closed categories [99], display map categories [110] and many other approaches to the categorical formulation of dependent types, see [55] for more information. Although in this thesis we consider, in essence, only the semantics of dependent types in locally cartesian closed categories, comprehension categories are a convenient way of dealing with technicalities, such as that substitution should be defined up to equality and not just isomorphism.

Definition 2.3.1 (Comprehension Category)**.** A *comprehension category* is a functor $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ such that $cod \circ \mathscr{P}\colon \mathbb{E} \to \mathbb{B}$ is a fibration and each cartesian map f with respect to that fibration is mapped by $\mathscr{P}$ to a pullback square. The comprehension category $\mathscr{P}$ is *full* if the functor $\mathscr{P}$ is full and faithful. It is called *split* if the fibration $cod \circ \mathscr{P}$ is split.

For the purposes of this thesis, we consider just the following sub-class of comprehension categories.

Definition 2.3.2 (Comprehension Category with Unit)**.** A *comprehension category with unit* is given by a fibration $p\colon \mathbb{E} \to \mathbb{B}$ with fibred terminal objects such that the terminal object functor $1\colon \mathbb{B} \to \mathbb{E}$ has a right adjoint $\{-\}\colon \mathbb{E} \to \mathbb{B}$. This data induces a comprehension category $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ by the mapping $\mathscr{P}\colon A \mapsto p(\varepsilon_A)$, where $\varepsilon_A\colon 1\{A\} \to A$ is the counit of $1 \dashv \{-\}$. The comprehension category with unit is called *full* if $\mathscr{P}$ is full and faithful.

Given an object B in $\mathbb{E}_\Gamma$, we use the notation $\pi_B\colon \{B\} \to \Gamma$ for the map $\mathscr{P}B$ in $\mathbb{B}$.

Definition 2.3.3 (Products, Sums)**.** A comprehension category with unit $p\colon \mathbb{E} \to \mathbb{B}$ has *products* (resp. *sums*) if, for each object A in $\mathbb{E}$, there is an adjunction $\pi_A^* \dashv \Pi_A$ (resp. $\Sigma_A \dashv \pi_A^*$) satisfying the Beck-Chevalley condition that, for each cartesian morphism $f\colon A \to B$, the canonical natural transformation $(pf)^*\Pi_B \to \Pi_A\{f\}^*$ (resp. $\Sigma_A\{f\}^* \to (pf)^*\Sigma_B$) is an isomorphism.

If in this situation p is a split fibration and the canonical natural transformation is the identity then we say that p has *split products* (resp. *split sums*).

Definition 2.3.4 (Strong Sums)**.** Sums in a comprehension category with unit $p\colon \mathbb{E} \to \mathbb{B}$ are *strong* if all the canonical injections κ in

$$\begin{array}{ccc} \{B\} & \xrightarrow[\cong]{\kappa} & \{\Sigma_A B\} \\ {\scriptstyle \pi_B}\downarrow & & \downarrow{\scriptstyle \pi_{\Sigma_A B}} \\ \{A\} & \xrightarrow[\pi_A]{} & \Gamma \end{array}$$

are isomorphisms.

Definition 2.3.5 (Closed Comprehension Category)**.** A *closed comprehension category* is a full comprehension category with unit that has products and strong sums as well as a terminal object in its base.

Examples of closed comprehension categories can be found in [56]. Most important for us is the fact that, for a locally cartesian closed category $\mathbb{B}$, the codomain fibration $cod\colon \mathbb{B}^{\to} \to \mathbb{B}$ is a closed comprehension category with unit.

2.4 Dependent Types

We give a coarse outline of how dependent types are modelled by closed comprehension categories. We refer to the literature, e.g. [56, 110, 105, 48, 53, 84], for more detailed descriptions.

Comprehension categories are a generalisation of display map categories [99, 110], in which dependent types are modelled as follows. Assume a category $\mathbb{B}$. The objects of $\mathbb{B}$ correspond to contexts of the dependent type theory. The morphisms in $\mathbb{B}$ correspond to simultaneous substitutions. A morphism $\Gamma \to \Delta$ in $\mathbb{B}$ represents a substitution using which one can go from a judgement in context Δ to a judgement in context Γ. The types in context $\Gamma \vdash A$ Type are modelled as morphisms $\pi_A\colon A \to \Gamma$ taken

from a distinguished class of *display maps* (in Seely's model in locally cartesian closed categories all maps are display maps). The idea is that the display map $A \to \Gamma$ represents the weakening substitution $(\Gamma, x\colon A) \to \Gamma$. Using this representation of types as display maps, terms can be modelled as sections, i.e. a term $\Gamma \vdash M : A$ corresponds to a map $M\colon \Gamma \to A$ in $\mathbb{B}$ for which $\pi_A \circ M = id$ holds. Intuitively, this means that the term M corresponds to a substitution $\Gamma \to (\Gamma, x\colon A)$ that is the identity on all variables but x. With this interpretation, substitution is modelled by pullback. The substitution of a type $\Gamma \vdash A$ Type along a map σ is given by the following pullback.

$$\begin{array}{ccc} \Gamma, A[\sigma] & \xrightarrow{\overline{\sigma}} & \Delta, A \\ {\scriptstyle \pi_{A[\sigma]}}\downarrow & & \downarrow{\scriptstyle \pi_A} \\ \Gamma & \xrightarrow[\sigma]{} & \Delta \end{array}$$

For substitution of terms, the universal property of the pullback gives, for any section M of π_A, a unique section $M[\sigma]$ of $\pi_{A[\sigma]}$ for which $\overline{\sigma} \circ M[\sigma] = M \circ \sigma$ holds. For this implementation of substitution to work, the category $\mathbb{B}$ must have pullbacks of display maps along arbitrary morphisms. Dependent sums and products are interpreted by left and right adjoints to pullback along a display map: $\Sigma_A \dashv \pi_A^* \dashv \Pi_A$. If all maps are display maps then the existence of these adjoints just amounts to local cartesian closure.

Comprehension categories are a generalisation of display map categories. Instead of identifying types and display maps, types are modelled as a separate entity that induces the display maps. In this way closed comprehension categories better formulate the concept that types are indexed over contexts. The fibration $p\colon \mathbb{E} \to \mathbb{B}$ captures how the types are indexed over contexts and how substitution is implemented. The base category $\mathbb{B}$ corresponds to contexts and substitutions as before. The total category $\mathbb{E}$ now consists of types and terms. The objects of $\mathbb{E}$ correspond to the types in context, i.e. an object A in $\mathbb{E}_\Gamma$ represents a type $\Gamma \vdash A$ Type. A morphism $A \to B$ in $\mathbb{E}_\Gamma$ represents a term $\Gamma, x\colon A \vdash M : B$. The comprehension describes how the types in $\mathbb{E}$ give rise to display maps. The display map for a type A in $\mathbb{E}_\Gamma$ is given by the map $\pi_A\colon \{A\} \to \Gamma$. The requirement in the definition of a comprehension category that cartesian morphisms are mapped to pullback squares means that the notion of substitution that is part of the fibration p is essentially the same as the pullback implementation of substitution in the induced display map interpretation.

In a closed comprehension category terms can be described equivalently in several ways. The equivalence of different descriptions is given by the following part of Lemma 10.4.9 of [56].

Lemma 2.4.1. *Let $p\colon \mathbb{E} \to \mathbb{B}$ be a comprehension category with unit. For all $u\colon \Gamma \to \Delta$ in $\mathbb{B}$ and all B in $\mathbb{E}_\Delta$ there is an isomorphism $\mathbb{B}/\Delta(u, \pi_B) \cong \mathbb{E}_\Gamma(1, u^*B)$ natural in u and B.*

By the display map interpretation, the term $\Gamma \vdash M : A$ amounts to a section of π_A, i.e. a map in $\mathbb{B}/\Gamma(id, \pi_A)$. By the lemma, the term also corresponds to a map $1_\Gamma \to A$ in $\mathbb{E}_\Gamma$. More generally, assume types $\Gamma \vdash A$ Type and $\Gamma \vdash B$ Type. Terms of the form $\Gamma, x\colon A \vdash M : B$ correspond to maps $1 \to \pi_A^* B$ in $\mathbb{E}_{\{A\}}$. By the lemma, this is the same as a map of type $\pi_A \to \pi_B$ in $\mathbb{B}/\Gamma$. Since the comprehension functor is, by assumption, full and faithful, this is the same as a map of type $A \to B$ in $\mathbb{E}_\Gamma$.

Depending on the situation, any of these equivalent descriptions of can be useful. We will use the most convenient one without further comment.

2.5 Monoidal Structure

In this thesis we study fibrations that have a monoidal base category. For comprehension categories with a monoidal structure $\otimes$ on their base we have useful notions of simple monoidal sums $\Sigma^{\otimes}$ and simple monoidal products $\Pi^{\otimes}$, whose definition is the purpose of this section. After recalling the definition of monoidal structure, we first describe $\Sigma^{\otimes}$ and $\Pi^{\otimes}$ in codomain fibrations and then generalise this definition to comprehension categories.

Definition 2.5.1. A *monoidal category* consists of a category $\mathbb{B}$, a functor $\otimes : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$, an object I in $\mathbb{B}$ and natural isomorphisms $l_A : I \otimes A \to A$, $r_A : A \otimes I \to A$, and $a_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ such that the following diagrams commute

$$\begin{array}{ccccc}
((A \otimes B) \otimes C) \otimes D & & \xrightarrow{\quad a \quad} & & (A \otimes B) \otimes (C \otimes D) \\
\downarrow{\scriptstyle a \otimes D} & & & & \downarrow{\scriptstyle a} \\
(A \otimes (B \otimes C)) \otimes D & \xrightarrow[a]{} & A \otimes ((B \otimes C) \otimes D) & \xrightarrow[A \otimes a]{} & A \otimes (B \otimes (C \otimes D))
\end{array}$$

$$\begin{array}{ccc}
(A \otimes I) \otimes B & \xrightarrow{\quad a \quad} & A \otimes (I \otimes B) \\
& {\scriptstyle r \otimes B} \searrow \quad \swarrow {\scriptstyle A \otimes l} & \\
& A \otimes B &
\end{array}$$

Definition 2.5.2. A *symmetric monoidal category* is a monoidal category with an additional natural transformation $s_{A,B} : A \otimes B \to B \otimes A$ making the following diagrams commute.

$$\begin{array}{ccccc}
(A \otimes B) \otimes C & \xrightarrow{a} & A \otimes (B \otimes C) & \xrightarrow{s} & (B \otimes C) \otimes A \\
\downarrow{\scriptstyle s \otimes C} & & & & \downarrow{\scriptstyle a} \\
(B \otimes A) \otimes C & \xrightarrow[a]{} & B \otimes (A \otimes C) & \xrightarrow[B \otimes s]{} & B \otimes (C \otimes A)
\end{array}$$

$$\begin{array}{ccc}
A \otimes I & \xrightarrow{\quad s \quad} & I \otimes A \\
& {\scriptstyle r} \searrow \quad \swarrow {\scriptstyle l} & \\
& A &
\end{array}
\qquad
\begin{array}{ccc}
A \otimes B & & \\
\downarrow{\scriptstyle s} & \searrow^{id} & \\
B \otimes A & \xrightarrow[s]{} & A \otimes B
\end{array}$$

Definition 2.5.3. A *monoidal closed category* is a monoidal category $\mathbb{B}$ in which, for each object A, the functor $- \otimes A : \mathbb{B} \to \mathbb{B}$ has right adjoint $A \multimap - : \mathbb{B} \to \mathbb{B}$.

Definition 2.5.4. An *affine monoidal category* is a monoidal category in which the unit I is terminal.

In an affine monoidal category there are (natural) maps $\dot{\pi}_1 : A \otimes B \to A$ and $\dot{\pi}_2 : A \otimes B \to B$ defined by

$$A \otimes B \xrightarrow{A \otimes !_B} A \otimes I \cong A \qquad A \otimes B \xrightarrow{!_A \otimes B} I \otimes B \cong B,$$

where $!_A : A \to I$ and $!_B : B \to I$ are the unique maps to the terminal object I.

Definition 2.5.5. A *strict affine monoidal category* is an affine monoidal category in which, for any two objects A and B, the maps $\dot{\pi}_1 : A \otimes B \to A$ and $\dot{\pi}_2 : A \otimes B \to B$ are jointly monomorphic in the sense that, for any two maps $f, g : C \to A \otimes B$ satisfying $\dot{\pi}_1 \circ f = \dot{\pi}_1 \circ g$ and $\dot{\pi}_2 \circ f = \dot{\pi}_2 \circ g$, we have $f = g$.

An affine monoidal category with binary products is a strict affine category if and only if, for all objects A and B, the maps $\langle \dot{\pi}_1, \dot{\pi}_2 \rangle : A \otimes B \rightarrowtail A \times B$ are monomorphic. We often write ι for $\langle \dot{\pi}_1, \dot{\pi}_2 \rangle$.

We will typically use the notation $(\otimes, \multimap)$ for a monoidal closed structure and $(*, -\!\!*)$ for a strict affine monoidal closed structure.

2.5.1 Monoidal Structure in a Codomain Fibration

We now consider the codomain fibration $cod : \mathbb{B}^{\to} \to \mathbb{B}$ for a monoidal category $\mathbb{B}$ with pullbacks, and show how multiplicative products $\Pi^{\otimes}$ and sums $\Sigma^{\otimes}$ can be defined for this fibration. This definition of $\Pi^{\otimes}$ and $\Sigma^{\otimes}$, which makes sense only in a codomain fibration, will be the basis of the general definition in the next section.

We first briefly recall *simple* products Π and *simple* sums Σ in a codomain fibration; see e.g. [44] for further details. Simple products and sums correspond to the types $\Pi x : A.B$ and $\Sigma x : A.B$ in which A is a simple, or closed, type. For an object A of $\mathbb{B}$ one considers the reindexing functor $\pi_A^* : \mathbb{B}/\Gamma \to \mathbb{B}/(\Gamma \times A)$ for the first projection $\pi_A : \Gamma \times A \to \Gamma$. This functor is a fibred functor from the codomain fibration $cod : \mathbb{B}^{\to} \to \mathbb{B}$ to the fibration $Gl(- \times A) : \mathbb{B}/(- \times A) \to \mathbb{B}$ defined by the following pullback of categories.

$$\begin{array}{ccc} \mathbb{B}/(- \times A) & \longrightarrow & \mathbb{B}^{\to} \\ {\scriptstyle Gl(-\times A)}\downarrow & \lrcorner & \downarrow{\scriptstyle cod} \\ \mathbb{B} & \xrightarrow[-\times A]{} & \mathbb{B} \end{array}$$

Since, for each $f : B \to \Gamma$, the square

$$\begin{array}{ccc} B \times A & \xrightarrow{\pi_A} & B \\ {\scriptstyle f\times A}\downarrow & \lrcorner & \downarrow{\scriptstyle f} \\ \Gamma \times A & \xrightarrow[\pi_A]{} & \Gamma \end{array}$$

is a pullback, the functor $\pi_A^* : \mathbb{B}/\Gamma \to \mathbb{B}/(\Gamma \times A)$ may be defined as taking an object $f : B \to \Gamma$ to the object $f \times A : B \times A \to \Gamma \times A$. Simple sums are defined as a fibred left adjoint $\Sigma_A : Gl(- \times A) \to cod$ to π_A^*. Simple products are defined as a fibred right adjoint $\Pi_A : Gl(- \times A) \to cod$ to π_A^*. The importance of having *fibred* adjunctions is that this includes the Beck-Chevalley conditions that the canonical natural

transformations

$$\Sigma_A(u \times A)^* \longrightarrow u^*\Sigma_A \qquad\qquad u^*\Pi_A \longrightarrow \Pi_A(u \times A)^*$$

are isomorphisms. The canonical natural transformations are given by the adjoint transposes of

$$(u \times A)^* \xrightarrow[(u*A)^*\eta]{} (u \times A)^*\pi_A\Sigma_A \xrightarrow[\cong]{} \pi_A^* u^*\Sigma_A$$

$$\pi_A^* u^*\Pi_A \xrightarrow[\cong]{} (u \times A)^*\pi_A\Pi_A \xrightarrow[(u*A)^*\varepsilon]{} (u \times A)^* \ .$$

Concretely, the functor Σ_A may be defined by post-composition, taking the object $f\colon B \to \Gamma \times A$ to the object $\pi_A \circ f\colon B \to \Gamma$. If $\mathbb{B}$ is cartesian closed then the functor Π_A may be defined as taking an object $f\colon B \to \Gamma \times A$ to the map $\Pi_A f$ in the following pullback.

$$\begin{array}{ccc} \Pi_A B & \longrightarrow & A \Rightarrow B \\ {\scriptstyle \Pi_A f}\downarrow & \lrcorner & \downarrow{\scriptstyle A \Rightarrow f} \\ \Gamma & \xrightarrow[\eta]{} & A \Rightarrow (\Gamma \times A) \end{array}$$

This description of simple sums and products can be generalised to simple monoidal sums and products simply by replacing the cartesian product $\times$ with the monoidal product $\otimes$. Instead of the reindexing functor π_A^*, we now consider the functor $W_A\colon \mathbb{B}/\Gamma \to \mathbb{B}/(\Gamma \otimes A)$, defined by taking $f\colon B \to \Gamma$ to $f \otimes A\colon B \otimes A \to \Gamma \otimes A$. We now assume that the functor $- \otimes A : \mathbb{B} \to \mathbb{B}$ preserves pullbacks, which is equivalent to W_A being a fibred functor of type

$$\begin{array}{ccc} \mathbb{B}^{\to} & \xrightarrow{W_A} & \mathbb{B}/(- \otimes A) \ . \\ & {\scriptstyle cod}\searrow \quad \swarrow{\scriptstyle Gl(-\otimes A)} & \\ & \mathbb{B} & \end{array}$$

With this definition, *simple monoidal sums* $\Sigma^\otimes$ and *simple monoidal products* $\Pi^\otimes$ can be defined as fibred left and right adjoints $\Sigma^\otimes\colon\ Gl(- \otimes A) \to cod$ and $\Pi^\otimes\colon\ Gl(- \otimes A) \to cod$ to W_A.

$$\begin{array}{ccc} & \overset{\Pi^\otimes}{\underset{\Sigma^\otimes}{\top}} & \\ \mathbb{B}^{\to} & \rightleftarrows & \mathbb{B}/(- \otimes A) \\ & {\scriptstyle cod}\searrow \quad \swarrow{\scriptstyle Gl(-\otimes A)} & \\ & \mathbb{B} & \end{array}$$

This definition of $\Sigma^\otimes$ and $\Pi^\otimes$ includes the Beck-Chevalley conditions that the following canonical natural transformations are isomorphisms.

$$\Sigma_A^\otimes(u \otimes A)^* \longrightarrow u^*\Sigma_A^\otimes \qquad\qquad u^*\Pi_A^\otimes \longrightarrow \Pi_A^\otimes(u \otimes A)^*$$

The definition of $\Sigma^\otimes$ and $\Pi^\otimes$ generalises that of Σ and Π, since if we let $\otimes$ be the cartesian product $\times$ then we have $W_A \cong \pi_A^*$, which implies $\Sigma^\times \cong \Sigma$ and $\Pi^\times \cong \Pi$ by uniqueness of adjoints.

By analogy with simple products, the functor $\Pi^{\otimes}$ can be constructed from a monoidal closed structure by taking an object $f\colon B \to \Gamma \otimes A$ to the map $\Pi^{\otimes}_A f$ in the following pullback:

$$\begin{array}{ccc} \Pi^{\otimes}_A B & \longrightarrow & A \multimap B \\ {\scriptstyle \Pi^{\otimes}_A f}\downarrow & \lrcorner & \downarrow{\scriptstyle A \multimap f} \\ \Gamma & \xrightarrow[\eta]{} & A \multimap (\Gamma \otimes A) \end{array} \tag{2.1}$$

Conversely, a monoidal closed structure can be defined from $\Pi^{\otimes}$:

Proposition 2.5.6. *The functor W_A has a fibred right adjoint $\Pi^{\otimes}_A$ if and only if $- \otimes A$ preserves pullbacks and has a right adjoint $A \multimap -$.*

Proof. The if-direction is defined by the pullback above. It is routine to verify that this defines a fibred right adjoint. It also follows from [60, Theorem 8].

For the only if-direction suppose that W_A has a fibred right adjoint $\Pi^{\otimes}_A$. That $- \otimes A$ preserves pullbacks follows because W_A is a fibred functor. For the construction of a right adjoint, we observe that the functor $- \otimes A$ can be written as the composite

$$\mathbb{B} \xrightarrow[-\otimes A]{} \mathbb{B} \qquad = \qquad \mathbb{B} \xrightarrow[\cong]{} \mathbb{B}/1 \xrightarrow[W_A]{} \mathbb{B}/(1 \otimes A) \xrightarrow[\Sigma_!]{} \mathbb{B}/1 \xrightarrow[\cong]{} \mathbb{B},$$

where $\Sigma_!$ is the functor that acts on objects by post-composition with the map $!\colon (1 \otimes A) \to 1$. The functor $\Sigma_!$ is left adjoint to reindexing $!^*$. Therefore, each functor in the composition has a right adjoint, which implies that $- \otimes A$ must have a right adjoint.

$$\mathbb{B} \underset{\cong}{\overset{\cong}{\underset{\top}{\leftarrow}}} \mathbb{B}/1 \underset{W_A}{\overset{\Pi^{\otimes}_A}{\underset{\top}{\leftarrow}}} \mathbb{B}/(1 \otimes A) \underset{\Sigma_!}{\overset{!^*}{\underset{\top}{\leftarrow}}} \mathbb{B}/1 \underset{\cong}{\overset{\cong}{\underset{\top}{\leftarrow}}} \mathbb{B}$$

□

In contrast to the additive case, multiplicative sums $\Sigma^{\otimes}$ do not come for free. In Corollary 3.4.33 we give an example of a codomain fibration that does not have all simple monoidal sums $\Sigma^{\otimes}$.

2.5.2 Monoidal Structure in a Comprehension Category

The definition of simple monoidal sums and products for the codomain fibration captures the essence of the structure we are interested in. However, to account for the bureaucracy of the syntax of type theory, we generalise this structure to full comprehension categories. This allows us to formulate simple monoidal sums and products for split fibrations, in which the interpretation of the syntax of type theory is much simpler than in non-split fibrations.

Perhaps the most obvious way of generalising simple monoidal sums and products from a codomain fibration $cod\colon \mathbb{B}^{\to} \to \mathbb{B}$ to a full comprehension category $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ is by assuming a fibred adjunction $\Sigma^{\otimes}_A \dashv W_A \dashv \Pi^{\otimes}_A$ between $(- \otimes A)^* p$ and p. In this adjunction, W_A should behave like the corresponding functor on the codomain fibration in the sense that $\pi_{W_A B}$ is isomorphic to $\pi_B \otimes A$ in $\mathbb{B}/(\Gamma \otimes A)$ and

this isomorphism is natural in B. Indeed, we make such a definition at the end of this section. However, the functor W_A can be problematic. For example, it is not clear how to extend the term model in Chapter 7 so that it has W_A. It turns out that $\Pi_A^\otimes$, at least, can be formulated without assuming W_A. This is what makes the completeness argument in Chapter 7 possible.

Simple monoidal products $\Pi_A^\otimes$ can be defined without assuming the functor W_A. This means $\Pi_A^\otimes$ can be defined without the assumption that, for all objects B in $\mathbb{E}_\Gamma$, there exists an object $W_A B$ in $\mathbb{E}_{\Gamma\otimes A}$ such that $\pi_{W_A B}$ is isomorphic to $\pi_B \otimes A$. If W_A exists, then we have $\mathbb{E}_{\Gamma\otimes A}(W_A B, C) \cong \mathbb{B}/(\Gamma\otimes A)(\pi_{W_A B}, \pi_C) \cong \mathbb{B}/(\Gamma\otimes A)(\pi_B \otimes A, \pi_C)$, since the comprehension is full and faithful. Hence, the adjunction $W_A \dashv \Pi_A^\otimes$ can be formulated by requiring a natural isomorphism $\mathbb{E}_\Gamma(B, \Pi_A^\otimes C) \cong \mathbb{B}/(\Gamma\otimes A)(\pi_B \otimes A, \pi_C)$, subject to a Beck-Chevalley condition. This definition, however, makes sense without the functor W_A. Of course, this definition without W_A need not be very useful, because there may not be enough morphisms in $\mathbb{E}$ mapped by the comprehension to morphisms in $\mathbb{B}/(\Gamma\otimes A)(\pi_B \otimes A, \pi_C)$, so that we could not use the isomorphism when working with the interpretation of type theory in the fibration. However, in the present case we have $\mathbb{B}/(\Gamma\otimes A)(\pi_B \otimes A, \pi_C) \cong \mathbb{E}_{\{B\}\otimes A}(1_{\{B\}\otimes A}, (\pi_B \otimes A)^* C)$ by Lemma 2.4.1. This means that the total category has enough structure to represent the whole of $\mathbb{B}/(\Gamma\otimes A)(\pi_B \otimes A, \pi_C)$. Therefore, we can define $\Pi_A^\otimes$ without the need for a functor W_A. The adjunction $W_A \dashv \Pi_A^\otimes$ amounts to a natural isomorphism $\mathbb{E}_\Gamma(B, \Pi_A^\otimes C) \cong \mathbb{E}_{\{B\}\otimes A}(1_{\{B\}\otimes A}, (\pi_B \otimes A)^* C)$.

For simple monoidal sums, this trick does not seem to work. For $\Sigma_A^\otimes$ we would have a natural isomorphism $\mathbb{E}_\Gamma(\Sigma_A^\otimes B, C) \cong \mathbb{B}/(\Gamma\otimes A)(\pi_B, \pi_C \otimes A)$, subject to a Beck-Chevalley condition. But now there appears to be no reason why we should be able to work with $\mathbb{B}/(\Gamma\otimes A)(\pi_B, \pi_C \otimes A)$ in the comprehension category.

In the rest of this section we first define simple monoidal products $\Pi^\otimes$ without W_A and then make precise the definition of W_A and $\Sigma_A^\otimes$ for full comprehension categories.

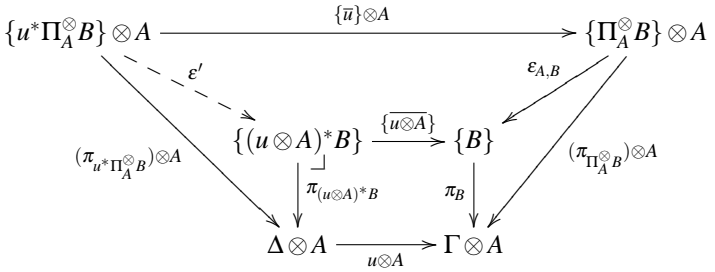
2.5.2.1 Simple Monoidal Products

Definition 2.5.7. Let $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^\to$ be a full comprehension category and $\otimes$ be a monoidal structure on $\mathbb{B}$. In such a comprehension category, *simple monoidal products* $\Pi^\otimes$ are given by the following data.

- For any two objects Γ and A in $\mathbb{B}$, a functor $\Pi_A^\otimes\colon \mathbb{E}_{\Gamma\otimes A} \to \mathbb{E}_\Gamma$.
- For all objects Γ and A in $\mathbb{B}$, all objects C in $\mathbb{E}_\Gamma$ and all objects B in $\mathbb{E}_{\Gamma\otimes A}$, an isomorphism $\mathbb{E}_\Gamma(C, \Pi_A^\otimes B) \cong \mathbb{B}/(\Gamma\otimes A)(\pi_C \otimes A, \pi_B)$ that is natural in C and B.
- The Beck-Chevalley condition, that for all maps u in $\mathbb{B}$ the canonical map $u^*\Pi_A^\otimes \to \Pi_A^\otimes (u\otimes A)^*$ is an isomorphism.

If the comprehension category is split and the canonical map $u^*\Pi_A^\otimes \to \Pi_A^\otimes (u\otimes A)^*$ is the identity then we speak of *split simple monoidal products* $\Pi^\otimes$.

We spell out the definition of the canonical morphism $u^*\Pi_A^\otimes \to \Pi_A^\otimes(u\otimes A)^*$. For B in $\mathbb{E}_{\Gamma\otimes A}$, let $\varepsilon_{A,B}\colon \pi_{\Pi_A^\otimes B}\otimes A \to \pi_B$ be the unique morphism in $\mathbb{B}/(\Gamma\otimes A)$ corresponding to the identity $\Pi_A^\otimes B \to \Pi_A^\otimes B$. Define the morphism $\varepsilon'\colon \pi_{u^*\Pi_A^\otimes B}\otimes A \to \pi_{(u\otimes A)^*B}$ to be the unique morphism making the diagram below commute.



The canonical morphism $u^*\Pi_A^\otimes B \to \Pi_A^\otimes(u\otimes A)^*B$ is the unique map corresponding to ε' under the isomorphism $\mathbb{E}_\Gamma(C,\Pi_A^\otimes B)\cong \mathbb{B}/(\Gamma\otimes A)(\pi_C\otimes A,\pi_B)$.

For split simple monoidal products the canonical map $u^*\Pi_A^\otimes B \to \Pi_A^\otimes(u\otimes A)^*B$ is the identity if and only if both $u^*\Pi_A^\otimes B = \Pi_A^\otimes(u\otimes A)^*B$ and $\varepsilon_{A,B}\circ(\{\overline{u}\}\otimes A) = \{\overline{u\otimes A}\}\circ\varepsilon_{A,(u\otimes A)^*B}$ hold for all u in $\mathbb{B}$ and B in $\mathbb{E}_{cod(u)\otimes A}$.

For the interpretation of the type theory, we need substitution equations for the term constructors of split simple monoidal products. We spell out these equations for reference.

Lemma 2.5.8. *The following are equivalent.*

1. *For all objects Γ and A in $\mathbb{B}$, all objects C in $\mathbb{E}_\Gamma$ and all objects B in $\mathbb{E}_{\Gamma\otimes A}$, an isomorphism $\mathbb{E}_\Gamma(C,\Pi_A^\otimes B)\cong\mathbb{B}/(\Gamma\otimes A)(\pi_C\otimes A,\pi_B)$ that is natural in C and B.*
2. *For all objects Γ and A in $\mathbb{B}$ and all objects B in $\mathbb{E}_{\Gamma\otimes A}$, a morphism $\varepsilon_{A,B}\colon \pi_{\Pi_A^\otimes B}\otimes A\to\pi_B$ in $\mathbb{B}/(\Gamma\otimes A)$ such that $\varepsilon_{A,B}$ is natural in B and, for each morphism $m\colon \pi_C\otimes A\to\pi_B$ in $\mathbb{B}/(\Gamma\otimes A)$, there exists a unique morphism $m^\flat\colon C\to\Pi_A^\otimes B$ in $\mathbb{E}_\Gamma$ satisfying $\varepsilon_{A,B}\circ(\{m^\flat\}\otimes A)=m$.*

Proof. For $1\Rightarrow 2$ take $\varepsilon_{A,B}$ as in the definition of the canonical map above. The existence of $m^\flat$ is immediate from the natural isomorphism $\mathbb{E}_\Gamma(C,\Pi_A^\otimes B)\cong\mathbb{B}/(\Gamma\otimes A)(\pi_C\otimes A,\pi_B)$, since both $\varepsilon_{A,B}\circ(\{m^\flat\}\otimes A)$ and m must correspond to the same map.

For $2\Rightarrow 1$ define a mapping from $\mathbb{E}_\Gamma(C,\Pi_A^\otimes B)$ to $\mathbb{B}/(\Gamma\otimes A)(\pi_C\otimes A,\pi_B)$ by assigning $n\colon C\to\Pi_A^\otimes B$ to $\varepsilon_{A,B}\circ(\{n\}\otimes A)$. The universality of ε makes it evident that this defines an isomorphism. Naturality follows from that of $\varepsilon_{A,B}$. □

Lemma 2.5.9. *Let $\mathscr{P}$ be a split full comprehension category with split simple monoidal products $\Pi^\otimes$. If, for any morphism $M\colon 1_{\Gamma\otimes A}\to B$ in $\mathbb{E}_{\Gamma\otimes A}$, we write $\lambda^\otimes M\colon 1_\Gamma\to\Pi_A^\otimes B$ for the unique morphism in $\mathbb{E}_\Gamma$ satisfying $\varepsilon_{A,B}\circ(\{\lambda^\otimes M\}\otimes A)=\{M\}$, then we have $u^*(\lambda^\otimes M)=\lambda^\otimes(u\otimes A)^*M$ for all $u\colon\Delta\to\Gamma$ in $\mathbb{B}$.*

Proof. We have $u^*(\lambda^\otimes M)\colon 1_\Delta\to u^*\Pi_A^\otimes B=\Pi_A^\otimes(u\otimes A)^*B$ and $\lambda^\otimes(u\otimes A)^*M\colon 1_\Delta\to\Pi_A^\otimes(u\otimes A)^*B$. In the diagram below, the topmost square commutes because the comprehension category maps the carte-

sian map $\overline{u}$ to a pullback and $- \otimes A$ is functorial. The middle square commutes because of the Beck-Chevalley condition. The lowermost square commutes because it is the image of $\overline{u \otimes A}$ under comprehension.

$$\begin{array}{ccc}
\Delta \otimes A & \xrightarrow{u \otimes A} & \Gamma \otimes A \\
{\scriptstyle \{u^* \lambda^{\otimes} M\} \otimes A}\downarrow & & \downarrow{\scriptstyle \{\lambda^{\otimes} M\} \otimes A} \\
\{\Pi_A^{\otimes} (u \otimes A)^* B\} \otimes A & \xrightarrow{\{\overline{u}\} \otimes A} & \{\Pi_A^{\otimes} B\} \otimes A \\
{\scriptstyle \varepsilon_{A,(u \otimes A)^* B}}\downarrow & & \downarrow{\scriptstyle \varepsilon_{A,B}} \\
\{(u \otimes A)^* B\} & \xrightarrow{\{\overline{u \otimes A}\}} & \{B\} \\
{\scriptstyle \pi_{(u \otimes A)^* B}}\downarrow & & \downarrow{\scriptstyle \pi_B} \\
\Delta \otimes A & \xrightarrow[u \otimes A]{} & \Gamma \otimes A
\end{array} \tag{2.2}$$

Since $\{u^* \lambda^{\otimes} M\}$ is a section of $\pi_{u^* \Pi_A^{\otimes} B} = \pi_{\Pi_A^{\otimes} (u \otimes A)^* B}$ and $\varepsilon_{A,(u \otimes A)^* B}$ is a map from $\pi_{\Pi_A^{\otimes} (u \otimes A)^* B} \otimes A$ to $\pi_{(u \otimes A)^* B}$, it follows that the composite of the left column is the identity. Since $\{M\} = \varepsilon_{A,B} \circ (\{\lambda^{\otimes} M\} \otimes A)$ holds by definition, and substitution is given by pullback, we can use the pullback property of the lowermost square to obtain $\{(u \otimes A)^* M\} = \varepsilon_{A,(u \otimes A)^* B} \circ (\{u^* \lambda^{\otimes} M\} \otimes A)$. But since $\lambda^{\otimes} (u \otimes A)^* M$ is defined as the unique map satisfying $\{(u \otimes A)^* M\} = \varepsilon_{A,(u \otimes A)^* B} \circ (\{\lambda^{\otimes} (u \otimes A)^* M\} \otimes A)$, this implies the required $\lambda^{\otimes} (u \otimes A)^* M = u^* \lambda^{\otimes} M$. □

Lemma 2.5.10. *Let $\mathscr{P}$ be a split full comprehension category with split simple monoidal products. If, for any morphism $M \colon 1_\Gamma \to \Pi_A^{\otimes} B$ in $\mathbb{E}_\Gamma$, we write $\mathrm{app}_B^{\otimes} M \colon 1_{\Gamma \otimes A} \to B$ for the unique morphism in $\mathbb{E}_{\Gamma \otimes A}$ satisfying $\varepsilon_{A,B} \circ (\{M\} \otimes A) = \{\mathrm{app}^{\otimes} M\}$, then we have $(u \otimes A)^* (\mathrm{app}_B^{\otimes} M) = \mathrm{app}^{\otimes}_{(u \otimes A)^* B} (u^* M)$ for all morphisms $u \colon \Delta \to \Gamma$ in $\mathbb{B}$.*

In this lemma, the morphism $\mathrm{app}_B^{\otimes} M$ is uniquely determined because the functor $\mathscr{P}$ is full and faithful. We will often write just $\mathrm{app}^{\otimes} M$ for $\mathrm{app}_B^{\otimes} M$ if the subscript is clear from the context.

Proof. Let $M \colon 1_\Gamma \to \Pi_A^{\otimes} B$. Because we have $\{\mathrm{app}_N^{\otimes} M\} = \varepsilon_{A,B} \circ (\{M\} \otimes A)$ and $\{\mathrm{app}^{\otimes}_{(u \otimes A)^* B} (u^* M)\} = \varepsilon_{A,(u \otimes A)^* B} \circ (\{u^* M\} \otimes A)$, we can use the pullback square in diagram (2.2) to get $\{(u \otimes A)^* (\mathrm{app}_B^{\otimes} M)\} = \{\mathrm{app}^{\otimes}_{(u \otimes A)^* B} (u^* M)\}$. The required equality follows because $\mathscr{P}$ is full and faithful. □

In the rest of this chapter, we use the notation $\lambda^{\otimes}$ and $\mathrm{app}^{\otimes}$ from the previous two lemmas.

Lemma 2.5.11. *In a split comprehension category with split simple monoidal products, the following equalities hold for all $M \colon 1_{\Gamma \otimes A} \to B$ in $\mathbb{E}_{\Gamma \otimes A}$ and $N \colon 1_\Gamma \to \Pi_A^{\otimes} B$.*

$$\mathrm{app}^{\otimes} (\lambda^{\otimes} M) = M \qquad\qquad \lambda^{\otimes} (\mathrm{app}^{\otimes} N) = N$$

Proof. Straightforward unfolding of definitions. □

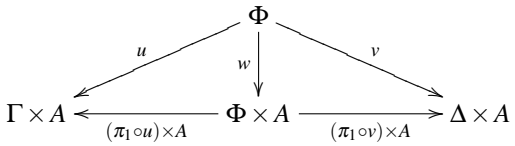
To simplify the syntax of the type theory, we will use a stronger Beck-Chevalley condition, as given by the following definition.

Definition 2.5.12. Let $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ be a split full comprehension category and $\otimes$ be an affine monoidal structure on $\mathbb{B}$. In such a comprehension category, *strong split simple monoidal products* $\Pi^{\otimes}$ are split simple monoidal products that in addition satisfy the following stronger Beck-Chevalley condition.

For all objects B in $\mathbb{E}_{\Gamma\otimes A}$ and D in $\mathbb{E}_{\Delta\otimes A}$ and all maps $M\colon 1_\Gamma \to \Pi^{\otimes}_A B$ in $\mathbb{E}_\Gamma$, $N\colon 1_\Delta \to \Pi^{\otimes}_A D$ in $\mathbb{E}_\Delta$, $u\colon \Phi \to \Gamma\otimes A$ in $\mathbb{B}$ and $v\colon \Phi \to \Delta\otimes A$ in $\mathbb{B}$ that satisfy $u^*B = v^*D$, $(\dot\pi_1 \circ u)^*M = (\dot\pi_1 \circ v)^*N$ and $\dot\pi_2 \circ u = \dot\pi_2 \circ v$, we have $u^*(\mathrm{app}^{\otimes}M) = v^*(\mathrm{app}^{\otimes}N)$.

The above condition includes $\mathrm{app}^{\otimes}(u^*M) = (u\otimes A)^*\mathrm{app}^{\otimes}M$ as a special case.

In the case where $\otimes$ is the cartesian product $\times$, simple products $\Pi^{\times}$ can be shown to be strong if the functor $\Pi^{\times}$ is injective on objects, meaning that $\Pi^{\times}_A B = \Pi^{\times}_A D$ implies $B = D$[1]. To see this observe that, under the assumption of the definition, the maps u and v can be factored as in the diagram below, in which $w = \langle \Phi, \pi_2 \circ u\rangle = \langle \Phi, \pi_2 \circ v\rangle$.

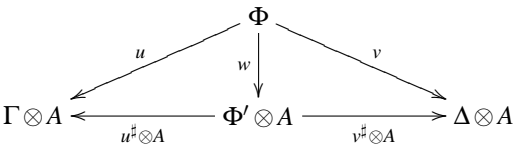


From the assumption $(\pi_1 \circ u)^*M = (\pi_1 \circ v)^*N$ it follows that $(\pi_1 \circ u)^*(\Pi^{\times}_A B) = (\pi_1 \circ v)^*(\Pi^{\times}_A D)$ holds. Hence, we obtain $((\pi_1 \circ u) \times A)^*B = ((\pi_1 \circ v) \times A)^*D$ using the Beck-Chevalley condition and injectivity of $\Pi^{\times}$. With this we have:

$$
\begin{aligned}
u^*(\mathrm{app}^{\times}_B M) &= w^*((\pi_1 \circ u) \times A)^*(\mathrm{app}^{\times}_B M) && \text{by above diagram}\\
&= w^*(\mathrm{app}^{\times}_{((\pi_1 \circ u)\times A)^*B}((\pi_1 \circ u)^*M)) && \text{by Beck-Chevalley condition}\\
&= w^*(\mathrm{app}^{\times}_{((\pi_1 \circ v)\times A)^*D}((\pi_1 \circ v)^*N)) && \text{by assumption}\\
&= w^*((\pi_1 \circ v) \times A)^*(\mathrm{app}^{\times}_D N) && \text{by Beck-Chevalley condition}\\
&= v^*(\mathrm{app}^{\times}_D N) && \text{by above diagram}
\end{aligned}
$$

In the case where $\otimes$ is not a cartesian product, this argument does of course not work anymore, since the above factorisation makes essential use of the diagonal map. Interestingly, however, simple monoidal sums $\Sigma^{\otimes}$ can be used to obtain a similar factorisation that does not use the diagonal map.

Lemma 2.5.13. *For any two maps $u\colon \Phi \to \Gamma\otimes A$ and $v\colon \Phi \to \Delta\otimes A$ in $\mathbb{B}$ satisfying $\dot\pi_2 \circ u = \dot\pi_2 \circ v$, there exist maps w, $u^\sharp$ and $v^\sharp$ such that the diagram below commutes.*

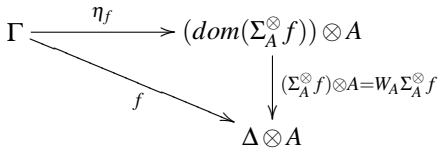


[1] This injectivity assumption should be inessential. It appears that we could change the definition of strong simple monoidal products to include the additional assumption $((\dot\pi_1 \circ u)\otimes A)^*B = ((\dot\pi_1 \circ v)\otimes A)^*D$ without essentially affecting the results in this thesis. Although this would make the injectivity assumption here unnecessary, it would also require a technical reworking throughout all of this thesis. Since we feel anyway that the assumption of strength for $\Pi^{\otimes}$ is not ideal and should be replaced by a better definition, we leave it as it is for now.

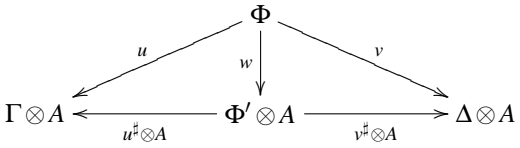
Proof. Let f be the object of $\mathbb{B}^{\to}$ given by $(!_\Gamma \otimes A) \circ u \colon \Phi \to 1 \otimes A$. Now define $\Phi' \stackrel{\text{def}}{=} dom(\Sigma_A^\otimes f)$ and $w \stackrel{\text{def}}{=} \eta_f$, where η is the unit of the fibred adjunction $\Sigma_A^\otimes \dashv W_A$. By definition of f, the morphism u defines a map from f to $(!_\Gamma \otimes A)$ in $\mathbb{B}/(1 \otimes A)$. Since $(!_\Gamma \otimes A)$ is the same as $W_A !_\Gamma$, this means that u is a map of type $f \to W_A !_\Gamma$ in $\mathbb{B}/(1 \otimes A)$. Taking the adjoint transpose of this map, we obtain $u^\sharp \colon \Sigma_A^\otimes f \to !_\Gamma$ in $\mathbb{B}/1$. The left triangle in the above diagram commutes by universality of η_f. Since the assumption $\dot{\pi}_2 \circ u = \dot{\pi}_2 \circ v$ implies $f = (!_\Gamma \otimes A) \circ v$, the same construction works for $v^\sharp$. □

Proposition 2.5.14. *Let $\mathscr{P} \colon \mathbb{E} \to \mathbb{B}^{\to}$ be a split full comprehension category with split simple monoidal products $\Pi^\otimes$. The split simple monoidal products are strong if $\otimes$ is affine and the following conditions are satisfied.*

- *The codomain fibration on $\mathbb{B}$ has simple monoidal sums $\Sigma^\otimes$.*
- *For all objects Γ and A in $\mathbb{B}$, and B and C in $\mathbb{E}_{\Gamma \otimes A}$, if $\Pi_A^\otimes B = \Pi_A^\otimes C$ holds then so does $B = C$.*
- *For all objects Γ and A in $\mathbb{B}$ and each morphism $f \colon \Gamma \to \Delta \otimes A$ in $\mathbb{B}$, the reindexing functor $(\dot{\pi}_1 \circ \eta_f)^* \colon \mathbb{E}_{dom(\Sigma_A^\otimes f)} \to \mathbb{E}_\Gamma$ is injective on objects and morphisms. Here, the map η_f is the counit of the adjunction $\Sigma_A^\otimes \dashv W_A$ on the codomain fibration, as in the diagram below.*



Proof. Assume the data given in the definition of strong simple monoidal products. By the above lemma we have morphisms w, $u^\sharp$ and $v^\sharp$ such that the following diagram commutes.



Moreover, w is η_f for some f.

By assumption we have $(\dot{\pi}_1 \circ u)^*(\Pi_A^\otimes B) = (\dot{\pi}_1 \circ u)^*(\Pi_A^\otimes D)$ and also that $\Pi_A^\otimes$ is injective. Using the above factorisation, we get $(\dot{\pi}_1 \circ (u^\sharp \otimes A) \circ w)^*(\Pi_A^\otimes B) = (\dot{\pi}_1 \circ (v^\sharp \otimes A) \circ w)^*(\Pi_A^\otimes D)$. By naturality of $\dot{\pi}_1$, this implies $(u^\sharp \circ \dot{\pi}_1 \circ w)^*(\Pi_A^\otimes B) = (v^\sharp \circ \dot{\pi}_1 \circ w)^*(\Pi_A^\otimes D)$. Since $p \colon \mathbb{E} \to \mathbb{B}$ is a split fibration, this is equivalent to $(\dot{\pi}_1 \circ w)^*(u^\sharp)^*(\Pi_A^\otimes B) = (\dot{\pi}_1 \circ w)^*(v^\sharp)^*(\Pi_A^\otimes D)$. Using the injectivity of $(\dot{\pi}_1 \circ w)^*$, this implies $(u^\sharp)^*(\Pi_A^\otimes B) = (v^\sharp)^*(\Pi_A^\otimes D)$. By definition of simple monoidal products, we have $(u^\sharp)^*(\Pi_A^\otimes B) = (\Pi_A^\otimes (u^\sharp \otimes A)^* B)$ and likewise for the other product, so that we obtain $(\Pi_A^\otimes (u^\sharp \otimes A)^* B) = (\Pi_A^\otimes (v^\sharp \otimes A)^* D)$. By injectivity of $\Pi^\otimes$, this implies $(u^\sharp \otimes A)^* B = (v^\sharp \otimes A)^* D$. Similarly, we obtain $(u^\sharp)^* M = (v^\sharp)^* N$. Using

this, we can mirror the reasoning above this proposition.

$$\begin{aligned}
u^*(\mathrm{app}^{\otimes}_B M) &= w^*(u^\sharp \otimes A)^*(\mathrm{app}^{\otimes}_B M) && \text{by above diagram}\\
&= w^*(\mathrm{app}^{\otimes}_{(u^\sharp\otimes A)^* B}((u^\sharp)^* M)) && \text{by Beck-Chevalley condition}\\
&= w^*(\mathrm{app}^{\otimes}_{(v^\sharp\otimes A)^* D}((v^\sharp)^* N))\\
&= w^*(v^\sharp \otimes A)^*(\mathrm{app}^{\otimes}_D N) && \text{by Beck-Chevalley condition}\\
&= v^*(\mathrm{app}^{\otimes}_D N) && \text{by above diagram}
\end{aligned}$$

□

2.5.2.2 Monoidal Weakening Structure

We now come to the definition of W_A and $\Sigma^{\otimes}_A$ in a full comprehension category. First we consider how W_A can be formulated in a full comprehension category. Let $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ be a full comprehension category with underlying fibration $p\colon \mathbb{E} \to \mathbb{B}$ and let A be an object of $\mathbb{B}$. Consider the fibration defined by change-of-base as in the diagram below

$$\begin{array}{ccc}
(-\otimes A)^*\mathbb{E} & \xrightarrow{K} & \mathbb{E}\\
{\scriptstyle (-\otimes A)^* p}\downarrow & & \downarrow{\scriptstyle p}\\
\mathbb{B} & \xrightarrow[(-\otimes A)]{} & \mathbb{B}
\end{array}$$

Then, W_A should be a fibred functor of type $p \to (-\otimes A)^* p$.

$$\begin{array}{ccc}
\mathbb{E} & \xrightarrow{W_A} & (-\otimes A)^*\mathbb{E}\\
& {\scriptstyle p}\searrow \quad \swarrow{\scriptstyle (-\otimes A)^* p} & \\
& \mathbb{B} &
\end{array}$$

Furthermore, on the morphisms given by comprehension, W_A should behave like it does in the codomain fibration. This means we should have a natural isomorphism $k_A\colon \mathscr{P} \circ K \circ W_A \cong W_A^{cod} \circ \mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ satisfying $cod(k_A) = id$, where W_A^{cod} denotes the codomain version of W_A. Spelt out, this means that, for each object B in $\mathbb{E}$, there is an isomorphism $k_{A,B}$ as in

$$\begin{array}{ccc}
\{KW_A B\} & \xrightarrow[\cong]{k_{A,B}} & \{B\}\otimes A\ ,\\
& {\scriptstyle \mathscr{P}(KW_A B)}\searrow \quad \swarrow{\scriptstyle \mathscr{P}(B)\otimes A} & \\
& (pB)\otimes A &
\end{array}$$

and this isomorphism is natural in B. Notice that the codomain of $\mathscr{P}(KW_A B)$ is indeed $(pB)\otimes A$, since $cod(\mathscr{P}(KW_A B)) = (p \circ K \circ W_A)B = ((-\otimes A)\circ(-\otimes A)^* p \circ W_A)B = ((-\otimes A)\circ p)B = (pB)\otimes A$. A concise way of stating these requirements is by requiring the fibred functor $K \circ W_A$ over $(-\otimes A)$ to be a map of comprehension categories [54, Definition 4.1.4] from $\mathscr{P}$ to itself.

Definition 2.5.15. Let $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ be a full comprehension category and $\otimes$ be a monoidal structure on $\mathbb{B}$. A *monoidal weakening structure* on the comprehension category is given by, for each object A in $\mathbb{B}$, a fibred functor $W_A\colon p \to (-\otimes A)^* p$ together with a natural isomorphism $k_A\colon \mathscr{P} \circ K \circ W_A \cong W_A^{cod} \circ \mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ satisfying $cod(k_A) = id$. If $\mathscr{P}$ is a split full comprehension category and W_A is a split fibred functor then we speak of a *split monoidal weakening structure*.

In abuse of notation, we will usually denote the functor $K \circ W_A\colon \mathbb{E} \to \mathbb{E}$ just by W_A. The two uses of W_A are disambiguated by the types of the functors, e.g. when writing $\{W_A B\}$ it is clear that we mean $\{KW_A B\}$, since $\{-\}$ is a functor of type $\mathbb{E} \to \mathbb{B}$.

Proposition 2.5.16. *In a full comprehension category with monoidal weakening structure, the functor $W_A\colon p \to (-\otimes A)^* p$ is determined up to vertical natural isomorphism.*

Proof. Assume two fibred functors $W_A, W_A'\colon p \to (-\otimes A)^* p$ and natural isomorphisms $k_A\colon \mathscr{P} \circ K \circ W_A \cong W_A^{cod} \circ \mathscr{P}$ and $k_A'\colon \mathscr{P} \circ K \circ W_A' \cong W_A^{cod} \circ \mathscr{P}$ satisfying $cod(k_A) = cod(k_A') = id$.

To show that there is a natural vertical isomorphism between W_A and W_A' it suffices to give a natural vertical isomorphism between $K \circ W_A$ and $K \circ W_A'$, since the fibration $(-\otimes A)^* p$ is defined by the change-of-base pullback and because $(-\otimes A)^* p \circ W_A = (-\otimes A)^* p \circ W_A'$ holds by assumption. From the assumptions, we get a natural isomorphism $k_A' \circ k_A^{-1}\colon \mathscr{P} \circ K \circ W_A' \cong \mathscr{P} \circ K \circ W_A\colon \mathbb{E} \to \mathbb{B}^{\to}$. Since $\mathscr{P}$ is full and faithful, this gives us a natural isomorphism $i\colon K \circ W_A \cong K \circ W_A'\colon \mathbb{E} \to \mathbb{E}$ such that $k_A' \circ k_A^{-1} = \mathscr{P}(i)$ holds. This isomorphism is vertical, since we have $p(i) = cod(\mathscr{P}(i)) = cod(k_A' \circ k_A^{-1}) = id$. □

We observe that, under certain assumptions, W_A enjoys a stronger substitution property than the usual Beck-Chevalley condition $(u \otimes A)^* W_A B = W_A u^* B$, which holds because W_A is a fibred functor.

Proposition 2.5.17. *Let $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ be a split full comprehension category with split monoidal weakening structure W_A with respect to an affine symmetric monoidal structure $\otimes$. Assume the following conditions.*

- *The codomain fibration on $\mathbb{B}$ has simple monoidal sums $\Sigma^{\otimes}$.*
- *For all objects Γ and A in $\mathbb{B}$ and each $f\colon \Gamma \to \Delta \otimes A$ in $\mathbb{B}$, the functor $(\dot{\pi}_1 \circ \eta_f)^*\colon \mathbb{E}_{dom(\Sigma_A^* f)} \to \mathbb{E}_\Gamma$ is injective on objects and morphisms. Here, η_f is as in Proposition 2.5.14.*

Then, for all objects A in $\mathbb{B}$, B in $\mathbb{E}_\Gamma$ and C in $\mathbb{E}_\Delta$ and all maps $u\colon \Phi \to \Gamma \otimes A$ and $v\colon \Phi \to \Delta \otimes A$ in $\mathbb{B}$ such that $(\dot{\pi}_1 \circ u)^* B = (\dot{\pi}_1 \circ v)^* C$ and $\dot{\pi}_2 \circ u = \dot{\pi}_2 \circ v$ hold, we have $u^* W_A B = v^* W_A C$.

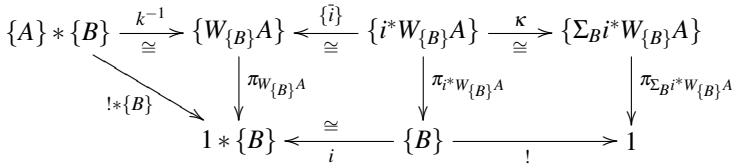
Proof. Lemma 2.5.13 provides a factorisation $u = (u^\sharp \otimes A) \circ w$ and $v = (v^\sharp \otimes A) \circ w$, where w is η_f for some f in $\mathbb{B}^{\to}$. From $(\dot{\pi}_1 \circ u)^* B = (\dot{\pi}_1 \circ v)^* C$, we obtain $(\dot{\pi}_1 \circ (u^\sharp \otimes A) \circ w)^* B = (\dot{\pi}_1 \circ (v^\sharp \otimes A) \circ w)^* B$. Using $(\dot{\pi}_1 \circ (u^\sharp \otimes A) \circ w)^* B = (u^\sharp \circ \dot{\pi}_1 \circ w)^* B = (\dot{\pi}_1 \circ w)^* (u^\sharp)^* B$ and a similar equation for C, this gives $(\dot{\pi}_1 \circ w)^* (u^\sharp)^* B = (\dot{\pi}_1 \circ w)^* (v^\sharp)^* C$. Since $w = \eta_f$, the injectivity assumption implies $(u^\sharp)^* B = (v^\sharp)^* C$. Since W_A is a split fibred functor, we have $(u^\sharp \otimes A)^* W_A B = W_A (u^\sharp)^* B$ and $(v^\sharp \otimes A)^* W_A C = W_A (v^\sharp)^* C$.

With $(u^\sharp)^*B = (v^\sharp)^*C$ we obtain $(u^\sharp \otimes A)^* W_A B = (v^\sharp \otimes A)^* W_A C$. But this implies $w^*(u^\sharp \otimes A)^* W_A B = w^*(v^\sharp \otimes A)^* W_A C$ and thus the required $u^* W_A B = v^* W_A C$. □

Finally, we show that W_A allows us to lift the monoidal structure $\otimes$ on $\mathbb{B}$ to $\mathbb{E}_1$. This generalises the situation in the codomain fibration, where such a lifting is available by means of $\mathbb{B} \cong \mathbb{B}/1$.

Proposition 2.5.18. *In any closed comprehension category with monoidal weakening structure, there exists a monoidal structure* $\overline{*}\colon \mathbb{E}_1 \times \mathbb{E}_1 \to \mathbb{E}_1$ *and, for any two objects A and B in* $\mathbb{E}_1$*, an isomorphism* $l_{A,B}\colon \{A\} * \{B\} \to \{A \overline{*} B\}$ *that is natural in both A and B.*

Proof. For objects A and B in $\mathbb{E}_1$, define the object $A \overline{*} B$ in $\mathbb{E}_1$ to be $\Sigma_B i^* W_{\{B\}} A$, where $i\colon \{B\} \to 1 * \{B\}$ is the canonical isomorphism. Consider the following commuting diagram



in which left triangle comes by definition of W_A, the centre square commutes by definition of comprehension categories and the right square comes from the definition of strong sums that are part of a closed comprehension category. The top row in this diagram gives the required isomorphism $l_{A,B}$.

It remains to define the morphism action of $\overline{*}$. Note first that the comprehension $\{-\}\colon \mathbb{E}_1 \to \mathbb{B}$ is full and faithful, since the comprehension category $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ is a full and faithful functor that maps $\mathbb{E}_1$ to $\mathbb{B}/1$. Using the isomorphisms $l_{A,B}$ and the fact that $\{-\}\colon \mathbb{E}_1 \to \mathbb{B}$ is full and faithful, we can lift the morphism action of $*$ to $\overline{*}$. For $f\colon A \to A'$ and $g\colon B \to B'$, the morphism $f \overline{*} g$ is defined as the unique morphism satisfying $\{f \overline{*} g\} = l_{A',B'} \circ (\{f\} * \{g\}) \circ l^{-1}_{A,B}$. That this defines a strict affine monoidal structure follows from the fact that $*$ is such a structure. □

2.5.2.3 Simple Monoidal Sums

Having defined W_A, we can now define $\Sigma^\otimes$ as a fibred left adjoint of W_A. Furthermore, in presence the of W_A, our earlier definition of $\Pi^\otimes_A$ can be simplified to a fibred adjunction $W_A \dashv \Pi^\otimes_A$.

Definition 2.5.19. Let $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$ be a comprehension category with monoidal weakening structure W_A. The comprehension category has *simple monoidal sums* if W_A has a fibred left adjoint $\Sigma^\otimes_A$. It has *simple monoidal products* if W_A has a fibred right adjoint $\Pi^\otimes_A$.

When all the structure in this definition is split we speak of *split simple monoidal sums* and *split simple monoidal products*.

It should be clear from the discussion at the beginning of this section that this definition of simple monoidal products agrees with the earlier definition without W_A.

There is a natural $\Sigma^{\otimes}$-analogue to *strong* $\Pi^{\otimes}$-types. However, for our present purposes it is not necessary to make such a definition; see the discussion in Section 9.2. Furthermore, it may be the case that, due to the presence of W_A, the simple monoidal sums $\Sigma_A^{\otimes}$ automatically satisfy a condition similar to that for strong monoidal products $\Pi_A^{\otimes}$, perhaps along the lines of Proposition 2.5.17.

2.5.3 Discussion and Further Work

In the definition of simple monoidal products $\Pi^{\otimes}$ and sums $\Sigma^{\otimes}$ we have assumed a fibration with a monoidal structure on its base. From a categorical point of view it would be more natural to consider some kind of fibred monoidal structure. The reason for not considering such a fibred structure here is that our particular models with names, to be constructed in the next chapter, do not have this structure. We discuss this further in Section 4.5.

In further work, the definition of strong simple monoidal products $\Pi^{\otimes}$ should be clarified. At present, the only justification for this definition is that it simplifies the syntax of the type theory in Chapter 4. Proposition 2.5.14 gives some evidence that there may be a better way of understanding this definition.

Another point not spelled out here is how, for a non-split fibration with $\Pi_A^{\otimes}$, W_A and $\Sigma_A^{\otimes}$, we can obtain an equivalent split fibration with that structure. It is known that the fibred Yoneda Lemma [56, Lemma 5.2.4] can be used to turn any fibration into an equivalent split fibration. This construction is spelled out in [46], where it is also shown that sums and products can be turned into split versions. Using the same construction is should be possible to obtain split versions of $\Pi_A^{\otimes}$ and $\Sigma_A^{\otimes}$. It is, however, not immediately obvious how to obtain a split version of W_A. This can be illustrated using the splitting of a codomain fibration. In the split fibration arising from a codomain fibration using the construction in [46], the objects in the total category are maps $\pi_B \colon B \to \Gamma$ that come equipped with a choice of pullback $f^*\pi_B \colon f^*B \to \Delta$ of π_B along any map $f \colon \Delta \to \Gamma$. The obvious definition of W_A would be to send $f^*\pi_B \colon f^*B \to \Delta$ to $(f^*\pi_B) \otimes A \colon (f^*B) \otimes A \to \Delta \otimes A$. But this definition only gives us a choice of pullback of $\pi_B \otimes A$ along maps of the form $f \otimes A \colon \Delta \otimes A \to \Gamma \otimes A$. For the definition of W_A, we need choices of pullbacks along arbitrary maps $g \colon \Phi \to \Gamma \otimes A$. Is there a canonical way of obtaining such choices, so that W_A becomes a split fibred functor? We believe that Lemma 2.5.13 can be used to obtain a positive answer. The reason is that by this lemma g can be factored as $(g' \otimes A) \circ w_g$. Moreover, if in this way we factor any other map $f \colon \Phi \to \Delta \otimes A$ as $(f' \otimes A) \circ w_f$, then $w_g = w_f$ holds whenever $\dot{\pi}_2 \circ f = \dot{\pi}_2 \circ g$ does. What we do not know at present, however, is if it is possible to obtain *strong* monoidal products $\Pi_A^{\otimes}$ using this splitting. For the purposes of this thesis, we use a different splitting, see Section 3.5.

Chapter 3

Monoidal Models of Dependent Types

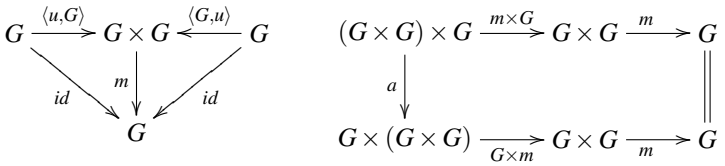
The handles we use to operate names and name-binding are a monoidal structure $*$, simple monoidal sums Σ^* and simple monoidal products Π^*. In this chapter we show what mechanism we wish to control with these handles. We introduce specific categories that will be used throughout the rest of this thesis as examples of categories with names. The purpose of this chapter is to define these categories and construct in them a monoidal structure $*$ as well as simple monoidal sums Σ^* and products Π^*. The prototypical category with this structure is the Schanuel topos, a categorical version of FM set theory. It forms the basis of the construction of other categories in this chapter. In particular, we show how the concepts from the Schanuel topos can be transferred to a realizability category to give a category with names in which the morphisms are computable functions. We present the Schanuel topos in such a way that the similarities between the constructions become clear.

3.1 Preliminaries

3.1.1 Group Actions

The categories constructed in this chapter are categories of group actions, which we briefly review here.

Definition 3.1.1. A *group object* in a category $\mathbb{B}$ with finite products is given by an object G and morphisms $u\colon 1 \to G$, $m\colon G \times G \to G$ and $i\colon G \to G$ making the following diagrams commute.



$$\begin{array}{ccccc} G\times G & \xrightarrow{i\times G} & G\times G & \xleftarrow{G\times i} & G\times G \\ \downarrow & & \downarrow{\scriptstyle m} & & \downarrow \\ 1 & \xrightarrow[u]{} & G & \xleftarrow[u]{} & 1 \end{array}$$

The group object is *commutative* if in addition the diagram below commutes.

$$\begin{array}{ccc} G\times G & \xrightarrow{\langle \pi_2,\pi_1\rangle} & G\times G \\ & {\scriptstyle m}\searrow & \downarrow{\scriptstyle m} \\ & & G\times G \end{array}$$

Definition 3.1.2. A *left action* of a group object G on an object A of $\mathbb{B}$ is a morphism $\mu\colon G\times A\to G$ for which the following diagrams commute.

$$\begin{array}{ccc} A & \xrightarrow{\langle u,A\rangle} & G\times A \\ & {\scriptstyle id}\searrow & \downarrow{\scriptstyle \mu} \\ & & A \end{array} \qquad \begin{array}{ccccc} G\times(G\times A) & \xrightarrow{G\times\mu} & G\times A & \xrightarrow{\mu} & A \\ \downarrow{\scriptstyle a} & & & & \| \\ (G\times G)\times A & \xrightarrow[m\times A]{} & G\times A & \xrightarrow[\mu]{} & A \end{array}$$

Definition 3.1.3. For any group object G in $\mathbb{B}$, the *category G-$\mathbb{B}$ of left G-actions in* $\mathbb{B}$ has as objects the pairs (A,μ_A) where A is an object of $\mathbb{B}$ and $\mu_A\colon G\times A\to A$ is a left action of G on A. The morphisms from (A,μ_A) to (B,μ_B) are the maps $f\colon A\to B$ of $\mathbb{B}$ that make the square below commute.

$$\begin{array}{ccc} G\times A & \xrightarrow{\mu_A} & A \\ {\scriptstyle G\times f}\downarrow & & \downarrow{\scriptstyle f} \\ G\times B & \xrightarrow[\mu_B]{} & B \end{array}$$

From now on we consider only left actions, and we will usually just say action.

There is an evident forgetful functor $U\colon G\text{-}\mathbb{B}\to\mathbb{B}$ mapping (A,μ_A) to the underlying object A.

Proposition 3.1.4. *For any group object G in $\mathbb{B}$, the forgetful functor $U\colon G\text{-}\mathbb{B}\to\mathbb{B}$ has a left adjoint F. If $\mathbb{B}$ is cartesian closed then U also has a right adjoint R.*

Proof. The left adjoint $F\colon\mathbb{B}\to G\text{-}\mathbb{B}$ maps A to the object $(G\times A,\mu)$ with the free action μ defined by $\mu = \ G\times(G\times A)\xrightarrow{\cong}(G\times G)\times A\xrightarrow{m\times A}G\times A$.

The right adjoint R maps an object A to $(G\Rightarrow A,\mu)$, where the action $\mu\colon G\times(G\Rightarrow A)\to(G\Rightarrow A)$ is defined in the internal language of the cartesian closed category $\mathbb{B}$ by $\mu(g,f)=\lambda g'.\,f(m(g',g))$. The verifications are routine. □

For any group object G in a category $\mathbb{B}$, the category G-$\mathbb{B}$ is by definition the same as the category of Eilenberg-Moore algebras for the monad arising from the adjunction $F\dashv U$ from Proposition 3.1.4. Explicitly, this monad is given by the functor $G\times-\colon\mathbb{B}\to\mathbb{B}$ with unit and multiplication defined by

$$\eta_A = \ A\xrightarrow{\langle u_G,A\rangle}G\times A\ , \qquad \mu_A = \ G\times(G\times A)\xrightarrow{a}(G\times G)\times A\xrightarrow{m\times A}G\times A\ .$$

Proposition 3.1.5. *For a category $\mathbb{B}$ with a group object G, the category G-$\mathbb{B}$ is equivalent to the category of Eilenberg-Moore algebras for the monad arising from the adjunction $F \dashv U$ from Proposition 3.1.4.*

If $\mathbb{B}$ is cartesian closed then we get an adjunction $U \dashv R$ from Proposition 3.1.4. The comonad arising from this adjunction is given by the functor $G \Rightarrow (-)\colon \mathbb{B} \to \mathbb{B}$ with counit defined by

$$\varepsilon_A = \; G \Rightarrow A \xrightarrow{\langle id, u_G \rangle} (G \Rightarrow A) \times G \xrightarrow{\text{ev}} A$$

and comultiplication $\delta_A\colon (G \Rightarrow A) \to (G \Rightarrow (G \Rightarrow A))$ defined to be the exponential transpose of

$$((G \Rightarrow A) \times G) \times G \xrightarrow{a^{-1}} (G \Rightarrow A) \times (G \times G) \xrightarrow{id \times m_G} (G \Rightarrow A) \times G \xrightarrow{\text{ev}} A \; .$$

Proposition 3.1.6. *For a cartesian closed category $\mathbb{B}$ with group object G, the category G-$\mathbb{B}$ is equivalent to the category of Eilenberg-Moore coalgebras for the comonad arising from the adjunction $U \dashv R$ from Proposition 3.1.4.*

Proof. This is an immediate consequence of Beck's theorem (Theorem 2.1.2). It is also straightforward to verify directly, since $G \Rightarrow -$ is right adjoint to $G \times -$, so that we have a one-to-one correspondence between maps $G \times A \to A$ and $A \to (G \Rightarrow A)$, which gives an equivalence of the two categories. □

3.1.2 Quasi-toposes

The categories constructed in this chapter are all quasi-toposes. For their construction, we find it convenient to work abstractly with the quasi-topos structure, and to appeal to general constructions with quasi-toposes. We stress, however, that we use quasi-toposes only for convenience in the construction of particular categories. We do not use the quasi-topos structure for any other purpose. In this section we fix the notation for quasi-toposes and state standard results; for more information see e.g. [117, 57, 64].

Definition 3.1.7. A *strong monomorphism* is a monomorphism $m\colon A \rightarrowtail B$ such that, for each commutative square as below in which e is epimorphic, there exists a unique diagonal d making the whole diagram commute.

$$\begin{array}{ccc} C & \xrightarrow{e} & D \\ \downarrow & \swarrow_{d} & \downarrow \\ A & \xrightarrow[m]{} & B \end{array}$$

All isomorphisms are strong monomorphisms, strong monomorphisms are closed under composition and they are closed under pullback along arbitrary maps [117, Proposition 10.3].

Definition 3.1.8. A *weak subobject classifier* is a map $true\colon 1 \to \Omega$ such that for each strong monomorphism $m\colon A \rightarrowtail B$ there exists a unique map $\chi\colon B \to \Omega$ making the following diagram a pullback.

$$\begin{array}{ccc} A & \xrightarrow{!} & 1 \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle true} \\ B & \xrightarrow[\chi]{} & \Omega \end{array}$$

Definition 3.1.9. A *quasi-topos* is a locally cartesian closed category with finite limits, finite colimits and a weak subobject classifier.

Definition 3.1.10. A *regular monomorphism* is a monomorphism that occurs as an equaliser.

Proposition 3.1.11. *In a quasi-topos, a monomorphism is regular if and only if it is strong.*

Proof. See e.g. [117, Proposition 12.5]. □

Definition 3.1.12. A category has *weak partial map classifiers* if, for each object A, there exist an object $A_\perp$ and a strong monomorphism $\eta_A\colon A \rightarrowtail A_\perp$ such that, for each strong monomorphism $m\colon B \rightarrowtail C$ and each map $f\colon B \to A$, there exists a unique map $\chi\colon C \to A_\perp$ making the diagram below a pullback.

$$\begin{array}{ccc} B & \xrightarrow{f} & A \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta_A} \\ C & \xrightarrow[\chi]{} & A_\perp \end{array}$$

Proposition 3.1.13. *Every quasi-topos has weak partial map classifiers.*

Proof. See e.g. [117, §19]. □

Proposition 3.1.14. *For any quasi-topos (respectively topos) $\mathbb{B}$ and any finite limit preserving comonad G on $\mathbb{B}$, the category of Eilenberg-Moore coalgebras for G is a quasi-topos (respectively topos).*

Proof. See e.g. [57, Theorem A4.2.4]. □

Corollary 3.1.15. *For any quasi-topos (respectively topos) $\mathbb{B}$ and any group object G in $\mathbb{B}$, the category G-$\mathbb{B}$ of G-actions on $\mathbb{B}$ is a quasi-topos (respectively topos).*

Proof. By Proposition 3.1.6 the category G-$\mathbb{B}$ is equivalent to the category of Eilenberg-Moore coalgebras for the comonad arising from the adjunction $U \dashv R$, where $U\colon G\text{-}\mathbb{B} \to \mathbb{B}$ is the forgetful functor. Since both U and R are right adjoints, this comonad preserves limits, so that we can apply Proposition 3.1.14 to get the required result. □

Corollary 3.1.16. *For any quasi-topos (respectively topos) $\mathbb{B}$ and any full and faithful functor $I\colon \mathbb{C} \hookrightarrow \mathbb{B}$ that preserves finite limits and has a right adjoint, the category $\mathbb{C}$ is a quasi-topos (respectively topos).*

Proof. Let $R\colon \mathbb{B} \to \mathbb{C}$ be a right adjoint to I. We use Beck's theorem (Theorem 2.1.2) to show that $\mathbb{C}$ is comonadic over $\mathbb{B}$ with comonad arising from $I \dashv R$. Since I has a right adjoint and since, being full and faithful functor, I reflects isomorphisms, it suffices to show that $\mathbb{C}$ has finite limits (which are by assumption preserved by I) to show that this theorem is applicable. Since I is full and faithful, the unit of the adjunction $I \dashv R$ is an isomorphism $Id \cong RI$. For a diagram $J\colon \mathbb{J} \to \mathbb{C}$ we therefore have $\lim J \cong \lim RIJ \cong R \lim IJ$, since R preserves limits. Hence, $\mathbb{C}$ inherits limits from $\mathbb{B}$. The proof is completed using Proposition 3.1.14. □

3.2 Constructing the Monoidal Closed Structure

In this section we give a specific construction of a monoidal exponent $-\!\!*$ for certain strict affine monoidal categories. In all the categories we consider in the rest of this chapter, the monoidal closed structure can be obtained using this construction. The construction generalises that of Menni [72, §4].

Throughout this section, we consider a quasi-topos $\mathbb{B}$ with a strict affine symmetric monoidal structure $*\colon \mathbb{B} \times \mathbb{B} \to \mathbb{B}$. We assume that A is an object of $\mathbb{B}$ for which the following conditions hold.

(C1) For all objects B, the canonical map $\iota_{B,A}\colon B * A \rightarrowtail B \times A$ is a strong monomorphism.

(C2) For each map $f\colon B * A \to C$, which uniquely determines map $\tilde{f}\colon B \times A \to C_\perp$ as in the pullback on the left, the square on the right is a pullback.

$$\begin{array}{ccc} B*A & \xrightarrow{f} & C \\ {\scriptstyle\iota}\downarrow & \lrcorner & \downarrow{\scriptstyle\eta_C} \\ B\times A & \xrightarrow[\tilde{f}]{} & C_\perp \end{array} \qquad \begin{array}{ccc} B*A & \xrightarrow{(\Lambda\tilde{f})*A} & (A \Rightarrow C_\perp)*A \\ {\scriptstyle\iota}\downarrow & \lrcorner & \downarrow{\scriptstyle\iota} \\ B\times A & \xrightarrow[(\Lambda\tilde{f})\times A]{} & (A \Rightarrow C_\perp)\times A \end{array}$$

Here, $\Lambda\tilde{f}$ denotes the exponential transpose of $\tilde{f}$.

(C3) The functor $(-)*A$ preserves exact sequences, that is coequaliser diagrams

$$B \underset{g}{\overset{f}{\rightrightarrows}} C \xrightarrow{h} D$$

in which f and g are a kernel pair of h.

Proposition 3.2.1. *For each object A, the functors $- \times A\colon \mathbb{B} \to \mathbb{B}$ and $- * A\colon \mathbb{B} \to \mathbb{B}$ preserve strong monomorphisms.*

Proof. That $- \times A$ preserves strong monomorphisms is a straightforward consequence of the fact that strong monomorphisms are closed under pullback. For $- * A$, consider the topmost square in the diagram below. We have to find a diagonal fill-in $d\colon Y \to B * A$ satisfying $(m * A) \circ d = g$ and $d \circ e = f$.

$$\begin{array}{ccc} X & \xrightarrow{e} & Y \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ B * A & \xrightarrow{m*A} & C * A \\ {\scriptstyle \iota_{B,A}}\downarrow & & \downarrow{\scriptstyle \iota_{C,A}} \\ B \times A & \xrightarrow{m \times A} & C \times A \end{array}$$

Since strong monomorphisms are closed under composition and are preserved by $- \times A$, we have that $(m \times A) \circ \iota_{B,A}$ is strong. Therefore, there exists a map $d\colon Y \to B * A$ satisfying $f = d \circ e$ and $(m \times A) \circ \iota_{B,A} \circ d = \iota_{C,A} \circ g$. Since the lower square commutes and $\iota_{C,A}$ is a mono, this implies $(m * A) \circ d = g$. Hence, d is the required fill-in. □

We come to the construction of a right adjoint $A \multimap (-)\colon \mathbb{B} \to \mathbb{B}$ to $(-) * A\colon \mathbb{B} \to \mathbb{B}$. First, let a and b be the characteristic maps defined by the weak partial map classifier in the diagrams below. Notice that the monomorphisms in these diagrams are all strong.

$$\begin{array}{ccc} (A \Rightarrow B_\perp) * A & \xrightarrow{\pi_2} & A \\ {\scriptstyle \iota}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta_A} \\ (A \Rightarrow B_\perp) \times A & \xrightarrow[a]{} & A_\perp \end{array} \qquad \begin{array}{ccc} B \times A & \xrightarrow{\pi_2} & A \\ {\scriptstyle \eta_B \times A}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta_A} \\ B_\perp \times A & \xrightarrow[b]{} & A_\perp \end{array}$$

Now define α and β by exponential transpose:

$$\alpha \stackrel{\text{def}}{=} \Lambda a : (A \Rightarrow B_\perp) \to (A \Rightarrow A_\perp)$$
$$\beta \stackrel{\text{def}}{=} \Lambda(b \circ \langle \text{ev}, \pi_2 \rangle) : (A \Rightarrow B_\perp) \to (A \Rightarrow A_\perp)$$

Here we write $\text{ev}\colon (A \Rightarrow B_\perp) \times A \to B_\perp$ for the application map. Define the object E and the monomorphism m as the equaliser of α and β.

$$E \xrightarrow[m]{} (A \Rightarrow B_\perp) \underset{\beta}{\overset{\alpha}{\rightrightarrows}} (A \Rightarrow A_\perp)\ .$$

Informally, the definitions of α and β read as

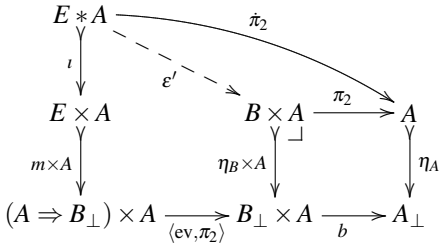
$$\alpha(f) \quad = \quad \lambda a. \begin{cases} a & \text{if } a \text{ is fresh for } f \\ \perp & \text{if } a \text{ is not fresh for } f \end{cases}$$
$$\beta(f) \quad = \quad \lambda a. \begin{cases} a & \text{if } f(a) \in B \\ \perp & \text{if } f(a) \notin B \end{cases}$$

Their equaliser therefore consists of all partial functions f in $A \Rightarrow B_\perp$ for which $f(a) \in B$ holds if and only if a is fresh for f. Indeed, in the internal language, E may be defined as

$$E = \{f\colon A \Rightarrow B_\perp \mid \forall a\colon A.\, f(a) \in B \iff f \# a\},$$

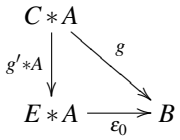
where we write $f \# a$ for the atomic proposition given by $\iota\colon (A \Rightarrow B_\perp) * A \rightarrowtail (A \Rightarrow B_\perp) \times A$.

We define a morphism $\varepsilon_0\colon E * A \to B$ by $\varepsilon_0 \stackrel{\text{def}}{=} \pi_1 \circ \varepsilon'$, where ε' is the unique morphism making the following diagram commute.



We can use the pullback since we have $\beta \circ m = \alpha \circ m$, which implies $b \circ \langle \mathrm{ev}, \pi_2 \rangle \circ (m \times A) = a \circ (m \times A)$ and further $b \circ \langle \mathrm{ev}, \pi_2 \rangle \circ (m \times A) \circ \iota_{E,A} = a \circ (m \times A) \circ \iota_{E,A} = a \circ \iota_{(A \Rightarrow B_\perp),A} \circ (m * A) = \eta_A \circ \pi_2$.

Lemma 3.2.2. *For each morphism $g\colon C * A \to B$ there exists a (not necessarily unique) morphism $g'\colon C \to E$ making the following diagram commute.*



Proof. Let $\tilde{g}\colon C \times A \to B_\perp$ be the classifying map of the partial map $(\iota_{C,A}, g)$, and let $\Lambda\tilde{g}\colon C \to (A \Rightarrow B_\perp)$ be its exponential transpose. To show the assertion, it suffices to show the existence of a map $g'\colon C \to E$ satisfying $m \circ g' = \Lambda\tilde{g}$, since we have:

$$\begin{array}{lrcll}
 & m \circ g' & = & \Lambda\tilde{g} & \\
\Longrightarrow & \mathrm{ev} \circ ((m \circ g') \times A) & = & \tilde{g} & \text{by universality of the exponent} \\
\Longrightarrow & \mathrm{ev} \circ ((m \circ g') \times A) \circ \iota_{C,A} & = & \tilde{g} \circ \iota_{C,A} & \text{by precomposition} \\
\Longrightarrow & \mathrm{ev} \circ (m \times A) \circ \iota_{E,A} \circ (g' * A) & = & \tilde{g} \circ \iota_{C,A} & \text{by naturality of } \iota \\
\Longrightarrow & \eta_B \circ \varepsilon_0 \circ (g' * A) & = & \tilde{g} \circ \iota_{C,A} & \text{by definition of } \varepsilon_0 \\
\Longrightarrow & \eta_B \circ \varepsilon_0 \circ (g' * A) & = & \eta_B \circ g & \text{by definition of } \tilde{g} \\
\Longrightarrow & \varepsilon_0 \circ (g' * A) & = & g & \text{since } \eta_B \text{ is monomorphic}
\end{array}$$

Since m is the equaliser of α and β, to show existence of g' satisfying $m \circ g' = \Lambda\tilde{g}$, it is enough to show that $\Lambda\tilde{g}$ equalises α and β.

To see that $\Lambda\tilde{g}$ equalises α and β, consider the following two diagrams.

$$\begin{array}{ccccc} C*A & \xrightarrow{\Lambda\tilde{g}*A} & (A \Rightarrow B_\perp)*A & \xrightarrow{\dot{\pi}_2} & A \\ {\scriptstyle \iota}\downarrow & \lrcorner & {\scriptstyle \iota}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta_A} \\ C\times A & \xrightarrow[\Lambda\tilde{g}\times A]{} & (A \Rightarrow B_\perp)\times A & \xrightarrow[a]{} & A_\perp \end{array} \tag{3.1}$$

$$\begin{array}{ccccc} C*A & \xrightarrow{\langle g,\dot{\pi}_2\rangle} & B\times A & \xrightarrow{\pi_2} & A \\ {\scriptstyle \iota}\downarrow & \lrcorner & {\scriptstyle \eta_B\times A}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta_A} \\ C\times A & \xrightarrow[\langle\tilde{g},\pi_2\rangle]{} & B_\perp\times A & \xrightarrow[b]{} & A_\perp \end{array} \tag{3.2}$$

In diagram (3.1), the right-hand square is a pullback by definition of a, and the left-hand square is a pullback by condition (C2). In diagram (3.2), the right-hand square is a pullback by definition of b, and the left-hand square is a pullback since in the following diagram the right-hand square is a pullback by the properties of products and the outermost square is a pullback by definition of $\tilde{g}$.

$$\begin{array}{ccccc} & & \overset{g}{\frown} & & \\ C*A & \xrightarrow[\langle g,\dot{\pi}_2\rangle]{} & B\times A & \xrightarrow[\pi_1]{} & B \\ {\scriptstyle \iota}\downarrow & & {\scriptstyle \eta_B\times A}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta_B} \\ C\times A & \xrightarrow[\langle\tilde{g},\pi_2\rangle]{} & B_\perp\times A & \xrightarrow[\pi_1]{} & B_\perp \\ & & \underset{\tilde{g}}{\smile} & & \end{array}$$

Notice that the top rows in both diagram (3.1) and diagram (3.2) are equal to $\dot{\pi}_2 \colon C*A \to A$. Therefore, we can use the uniqueness property of the weak partial map classifier η_A to get $a \circ ((\Lambda\tilde{g}) \times A) = b \circ \langle \tilde{g}, \pi_2 \rangle$. Now, we have $\langle \tilde{g}, \pi_2 \rangle = \langle \mathrm{ev}, \pi_2 \rangle \circ ((\Lambda\tilde{g}) \times A)$, using which we get $a \circ ((\Lambda\tilde{g}) \times A) = b \circ \langle \mathrm{ev}, \pi_2 \rangle \circ (\Lambda\tilde{g}) \times A$. Since α and β are the exponential transposes of a and $b \circ \langle \mathrm{ev}, \pi_2 \rangle$ respectively, this implies $\alpha \circ (\Lambda\tilde{g}) = \beta \circ (\Lambda\tilde{g})$. Therefore, $\Lambda\tilde{g}$ equalises α and β, which completes the proof. □

We remark that (C2) is used essentially in the proof of this lemma to show that $\Lambda\tilde{g} \colon C \to (A \Rightarrow B_\perp)$ factors through $m \colon E \to (A \Rightarrow B_\perp)$. In the internal language, the following sequent holds by definition of $\Lambda\tilde{g}$.

$$\forall a \colon A. \forall c \colon C.\ (\Lambda\tilde{g})(c)(a) \in B \iff c \# a.$$

Hence, to show that $\Lambda\tilde{g}$ factors through m, we have to show

$$\forall a \colon A. \forall c \colon C.\ c \# a \iff (\Lambda\tilde{g})(c) \# a,$$

which is precisely what condition (C2) amounts to.

To construct a right adjoint to $(-)*A$, we would like to strengthen the above lemma to the existence of a *unique* map g'. To get uniqueness, it is necessary to take a quotient of E. For the definition of the equivalence relation by which to take the quotient, we use the following lemma.

Lemma 3.2.3. *There exists an object S such that, for each morphism $f\colon B \to C$, there exists a morphism $s\colon B \to S$ making the following square a pullback.*

$$\begin{array}{ccc} B * A & \xrightarrow{\langle f,s\rangle * A} & (C \times S) * A \\ {\scriptstyle \iota}\downarrow & \lrcorner & \downarrow{\scriptstyle \iota} \\ B \times A & \xrightarrow[\langle f,s\rangle \times A]{} & (C \times S) \times A \end{array}$$

Proof. Let S be $(A \Rightarrow A_\perp)$. Let $\tilde{\dot{\pi}}_2\colon B \times A \to A_\perp$ be the classifier of the partial map $(\iota_{B,A}, \dot{\pi}_2)$, and let $s\colon B \to S$ be the exponential transpose of $\tilde{\dot{\pi}}_2$. By condition (C2), the following square is a pullback.

$$\begin{array}{ccc} B * A & \xrightarrow{s * A} & S * A \\ {\scriptstyle \iota}\downarrow & \lrcorner & \downarrow{\scriptstyle \iota} \\ B \times A & \xrightarrow[s \times A]{} & S \times A \end{array}$$

The assertion follows immediately. □

Let S be the object defined in the above lemma. Define e'_1 and e'_2 to be the classifying maps as in the following diagram, in which $i = 1, 2$.

$$\begin{array}{ccccc} (E \times E \times S) * A & \xrightarrow{\pi_i * A} & E * A & \xrightarrow{\varepsilon_0} & B \\ {\scriptstyle \iota}\downarrow & & & & \downarrow{\scriptstyle \eta_B} \\ (E \times E \times S) \times A & & \xrightarrow[e'_i]{} & & B_\perp \end{array}$$

Let $e_1 \stackrel{\text{def}}{=} \Lambda e'_1$ and $e_2 \stackrel{\text{def}}{=} \Lambda e'_2$, and define r_0 as their equaliser.

$$R_0 \xrightarrow[r_0]{} (E \times E \times S) \underset{e_2}{\overset{e_1}{\rightrightarrows}} (A \Rightarrow B_\perp)$$

Now define r as the image of $\pi_1 \circ r_0$, as in the following diagram.

$$\begin{array}{ccc} R_0 & \xrightarrow{r_e} & R \\ {\scriptstyle r_0}\downarrow & & \downarrow{\scriptstyle r} \\ E \times E \times S & \xrightarrow[\langle \pi_1, \pi_2\rangle]{} & E \times E \end{array}$$

Writing r_1 for $\pi_1 \circ r$ and r_2 for $\pi_2 \circ r$, define the object $(A \multimap B)$ by the following coequaliser.

$$R \underset{r_2}{\overset{r_1}{\rightrightarrows}} E \xrightarrow[c]{} (A \multimap B)$$

In the internal language, $r\colon R \rightarrowtail E \times E$ defines the following equivalence relation on E.

$$fRg \iff \exists s\colon S.\, \forall a\colon A.\, (\langle f, g, s\rangle \# a) \implies (\varepsilon_0(f)(a) = \varepsilon_0(g)(a))$$

In this formula, we write $\langle f, g, s\rangle \# a$ for the atomic proposition given by the strong monomorphism $\iota\colon (E \times E \times S) * A \rightarrowtail (E \times E \times S) \times A$.

Lemma 3.2.4. *The strong monomorphism $r\colon R \rightarrowtail E \times E$ is an equivalence relation.*

Proof. Reflexivity and symmetry are immediate. We check transitivity. We use the internal logic of the quasi-topos. We have to show $f\colon E, g\colon E, h\colon E \mid fRg, gRh \vdash fRh$. By the left rule for $\exists$, it suffices to show $f\colon E, g\colon E, h\colon E, s\colon S, s'\colon S \mid \varphi(f,g,s), \varphi(g,h,s') \vdash fRh$, where we write $\varphi(f,g,s)$ for $\forall a\colon A.\, (\langle f,g,s\rangle \# a) \implies (\varepsilon_0(f)(a) = \varepsilon_0(g)(a))$. By Lemma 3.2.3, there exists a morphism t of type $E \times E \times E \times S \times S \to S$ such that the following square is a pullback.

$$\begin{array}{ccc} (E \times E \times E \times S \times S) * A & \xrightarrow{\langle \pi_1,\pi_2,\pi_3,t\rangle * A} & (E \times E \times E \times S) * A \\ \downarrow{\scriptstyle \iota} & \lrcorner & \downarrow{\scriptstyle \iota} \\ (E \times E \times E \times S \times S) \times A & \xrightarrow[\langle \pi_1,\pi_2,\pi_3,t\rangle \times A]{} & (E \times E \times E \times S) \times A \end{array}$$

In the internal language, this pullback amounts to the sequent

$$f\colon E, g\colon E, h\colon E, s\colon S, s'\colon S, a\colon A \vdash \langle f,g,h,s,s'\rangle \# a \iff \langle f,g,h,t(f,g,h,s,s')\rangle \# a.$$

Using $\langle f,g,h,s,s'\rangle \# a \implies \langle f,g,s\rangle \# a$, we get $\langle f,g,h,t(f,g,h,s,s')\rangle \# a \implies \langle f,g,s\rangle \# a$. Similarly, we get $\langle f,g,h,t(f,g,h,s,s')\rangle \# a \implies \langle g,h,s'\rangle \# a$. We continue the above derivation, using the $\exists$-right rule with $t(f,g,h,s,s')$. Thus, it suffices to show

$$f\colon E, g\colon E, h\colon E, s\colon S, s'\colon S \mid \varphi(f,g,s), \varphi(g,h,s') \vdash \varphi(f,h,t(f,g,h,s,s')).$$

Using the rules for $\forall$ and $\implies$ on the right, it suffices to show

$$f,g,h\colon E, s,s'\colon S, a\colon A \mid \varphi(f,g,s), \varphi(g,h,s'), \langle f,g,h,t(f,g,h,s,s')\rangle \# a \vdash (\varepsilon_0(f)(a) = \varepsilon_0(h)(a)).$$

Using the above implications, showing the following sequent is enough.

$$f,g,h\colon E, s,s'\colon S, a\colon A \mid \varphi(f,g,s), \varphi(g,h,s'), \langle f,g,s\rangle \# a, \langle g,h,s'\rangle \# a \vdash (\varepsilon_0(f)(a) = \varepsilon_0(h)(a))$$

This sequent follows immediately by using $\forall$-left with a and transitivity of equality. □

This lemma shows that

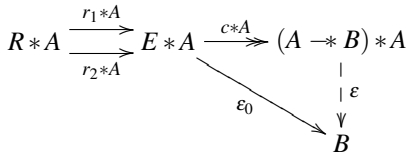
$$R \underset{r_2}{\overset{r_1}{\rightrightarrows}} E \xrightarrow[c]{} (A \multimap B)$$

is an exact sequence, since in a quasi-topos any strong equivalence relation is a kernel pair of some map [71, Prop. 4.3.3], and any such pair is also a kernel pair of its coequaliser [110, Lemma 5.6.6].

Proposition 3.2.5. *The definition of $A \multimap B$ extends to a functor $A \multimap -\colon \mathbb{B} \to \mathbb{B}$ right adjoint to $- * A\colon \mathbb{B} \to \mathbb{B}$.*

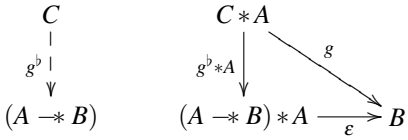
Proof. The proof generalises that of [72, Proposition 1.3]. By assumption (C3), the functor $- * A$ preserves exact sequences. Hence, $c * A$ is a coequaliser. We use its universal property to define the

counit $\varepsilon\colon (A \multimap B) * A \to B$ as the unique map making the diagram below commute.



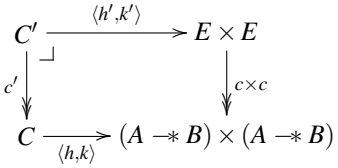
To show that the universal property of the coequaliser can indeed be used, we have to show $\varepsilon_0 \circ (r_1 * A) = \varepsilon_0 \circ (r_2 * A)$. Since $r \circ r_e$ is an image factorisation, the map r_e is a regular epi; hence r_e is the coequaliser of its kernel pair. Since $- * A$ preserves exact sequences, $r_e * A$ is also a regular epi. Hence, to show $\varepsilon_0 \circ (r_1 * A) = \varepsilon_0 \circ (r_2 * A)$ it suffices to show $\varepsilon_0 \circ ((\pi_1 \circ r_0) * A) = \varepsilon_0 \circ ((\pi_2 \circ r_0) * A)$. Since r_0 equalises e_1 and e_2, we have $e_1 \circ r_0 = e_2 \circ r_0$, which implies $\Lambda(e_1' \circ (r_0 \times A)) = \Lambda((e_2' \circ (r_0 \times A))$, by definition and naturality of the exponential adjunction, and further $e_1' \circ (r_0 \times A) = e_2' \circ (r_0 \times A)$, by the universal property of exponentials. This implies $e_1' \circ (r_0 \times A) \circ \iota = e_2' \circ (r_0 \times A) \circ \iota$ by precomposition. By naturality of ι and the definition of e_i', we get $\eta_B \circ \varepsilon_0 \circ ((\pi_1 \circ r_0) * A) = \eta_B \circ \varepsilon_0 \circ ((\pi_2 \circ r_0) * A)$. Since η_B is monomorphic, this implies the required $\varepsilon_0 \circ ((\pi_1 \circ r_0) * A) = \varepsilon_0 \circ ((\pi_2 \circ r_0) * A)$.

For universality of ε, we have to show that, for each map $g\colon C * A \to B$, there exists a unique map $g^\flat\colon C \to (A \multimap B)$ making the triangle on the right commute.

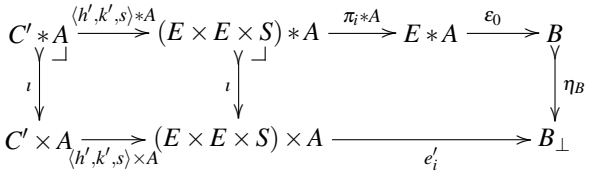


Existence follows from Lemma 3.2.2. By this lemma there exists $g'\colon C \to E$ satisfying $\varepsilon_0 \circ (g' * A) = g$. By definition of ε, this implies $\varepsilon \circ ((c \circ g') * A) = g$. Define $g^\flat \stackrel{\text{def}}{=} c \circ g'$.

For uniqueness, suppose there are maps $h, k\colon C \to (A \multimap B)$ satisfying $g = \varepsilon \circ (h * A) = \varepsilon \circ (k * A)$. Consider the pullback



Since this diagram commutes, we have $c \circ h' = h \circ c'$ and $c \circ k' = k \circ c'$. By functoriality, this implies $(c * A) \circ (h' * A) = (h * A) \circ (c' * A)$ and $(c * A) \circ (k' * A) = (k * A) \circ (c' * A)$. Since $\varepsilon \circ (h * A) = \varepsilon \circ (k * A)$, this implies $\varepsilon \circ (c * A) \circ (h' * A) = \varepsilon \circ (c * A) \circ (k' * A)$, i.e. $\varepsilon_0 \circ (h' * A) = \varepsilon_0 \circ (k' * A)$. By Lemma 3.2.3, there exists a map $s\colon C' \to S$ such that the square on the left below is a pullback. The square on the right below, in which $i = 1, 2$, is a pullback by construction of e_i.



By $\varepsilon_0 \circ (h' * A) = \varepsilon_0 \circ (k' * A)$, we have $\varepsilon_0 \circ (\pi_1 * A) \circ (\langle h', k', s\rangle * A) = \varepsilon_0 \circ (\pi_2 * A) \circ (\langle h', k', s\rangle * A)$. Hence the top row in the above diagram is the same for both $i = 1$ and $i = 2$. By the universal property of partial map classifiers, this implies $e'_1 \circ (\langle h', k', s\rangle \times A) = e'_2 \circ (\langle h', k', s\rangle \times A)$. By naturality of the exponential transpose, we obtain $(\Lambda e'_1) \circ \langle h', k', s\rangle = (\Lambda e'_2) \circ \langle h', k', s\rangle$, i.e. $e_1 \circ \langle h', k', s\rangle = e_2 \circ \langle h', k', s\rangle$. By the definition of r_0 as an equaliser of e_1 and e_2, the map $\langle h', k', s\rangle$ therefore factors through r_0. This implies that $\langle h', k'\rangle$ factors through r, i.e. there exists u with $\langle h', k'\rangle = r \circ u$. Hence, we have $h' = r_1 \circ u$ and $k' = r_2 \circ u$. Since c coequalises r_1 and r_2, this implies $c \circ h' = c \circ r_1 \circ u = c \circ r_2 \circ u = c \circ k'$. Since $h \circ c' = c \circ h'$ and $k \circ c' = c \circ k'$ hold and c' is epimorphic, this implies the required $h = k$. □

3.3 The Schanuel Topos

In this section we introduce the Schanuel topos, which is the prototypical example of a category with bindable names. The Schanuel topos corresponds to FM set theory in the same way the category of sets corresponds to ZF set theory. It may be thought of as a category of sets whose elements may somehow contain names (or atoms). FM set theory formalises the concept of 'somehow containing names' by equipping the sets with a name-permutation action. Following the presentation in [38, 87], we define the Schanuel topos as a category of group actions in **Sets**.

There are many equivalent ways of describing the Schanuel topos. It may be described: (i) as a category of continuous group actions; (ii) as a category of sheaves in $\mathbf{Sets}^{\mathbb{I}}$; (iii) as the classifying topos for an infinite decidable object; (iv) as a category of permutation algebras; and (v) as a category of named sets. For further information on these descriptions, we refer to [64, §III.9] and [57, A2.1.11(h)] for (i) and (ii); to [64, Exercise VIII.9] for (iii); and to [39] for (iv) and (v).

Let $\mathbf{N}$ be a countably infinite set, i.e. a set $\mathbf{N}$ that is isomorphic to the set of natural numbers. The elements of $\mathbf{N}$ will be called *names*. The group $Aut(\mathbf{N})$ is the group of automorphisms on $\mathbf{N}$, i.e. the group of bijective functions on $\mathbf{N}$ where the unit, multiplication and inverse are given by the identity function, function composition and inverse function respectively.

Definition 3.3.1. The set $\mathbf{N}$ is the underlying set of the object $(\mathbf{N}, \cdot_{\mathbf{N}})$ of the category $Aut(\mathbf{N})$-**Sets** where $\pi \cdot_{\mathbf{N}} n = \pi(n)$. We call this object the *object of names*, writing $\mathbf{N}$ for it.

Definition 3.3.2. Let $(A, \cdot_A)$ be an object of $Aut(\mathbf{N})$-**Sets** and let $a \in A$. A set $X \subseteq \mathbf{N}$ *supports* a if it satisfies $\forall \pi \in Aut(\mathbf{N}).\ (\forall x \in X.\ \pi(x) = x) \implies (\pi \cdot_A a = a)$. The element $a \in A$ is *finitely supported* if there exists a finite set supporting it. The object A has the *finite support property* if all its elements are finitely supported.

Proposition 3.3.3. *Let $(A, \cdot_A)$ be an object of $Aut(\mathbf{N})$-**Sets**. If $a \in A$ is finitely supported then there exists a finite set* $\mathrm{supp}(a) \subseteq \mathbf{N}$ *such that* $\mathrm{supp}(a) \subseteq X$ *holds for every set X supporting a.*

Proof. See [38, Proposition 3.4]. □

Definition 3.3.4. The *Schanuel topos* $\mathbb{S}$ is the full subcategory of $Aut(\mathbf{N})$-**Sets** consisting of all objects that have the finite support property.

In concrete terms, the objects of $\mathbb{S}$ are sets A with a left $Aut(\mathbf{N})$-action $\cdot_A : Aut(\mathbf{N}) \times A \to A$ such that $(A, \cdot_A)$ has the finite support property. A morphism $f : (A, \cdot_A) \to (B, \cdot_B)$ is a function $f : A \to B$ in **Sets** that is *equivariant*, meaning that $\forall \pi \in Aut(\mathbf{N}).\ \forall a \in A.\ \pi \cdot_B (f(a)) = f(\pi \cdot_A a)$ holds.

Proposition 3.3.5. *The following are true.*

1. *For all morphisms $f : A \to B$ in $\mathbb{S}$ and all $a \in A$, we have* $\mathrm{supp}(f(a)) \subseteq \mathrm{supp}(a)$.
2. *For all monomorphisms $m : A \rightarrowtail B$ in $\mathbb{S}$ and all $a \in A$, we have* $\mathrm{supp}(m(a)) = \mathrm{supp}(a)$.

Proof.

1. We need to show that any finite set $X \subseteq \mathbf{N}$ supporting an element $a \in A$ also supports $f(a)$. Let $a \in A$ and let X be a finite set of names supporting a. Let $\pi \in Aut(\mathbf{N})$ such that $\pi(n) = n$ holds for all $n \in X$. Then we have $\pi \cdot f(a) = f(\pi \cdot a) = f(a)$, by equivariance and since X supports a. Hence, X supports $f(a)$.
2. From the first point we already have $\mathrm{supp}(a) \supseteq \mathrm{supp}(m(a))$ for all $a \in A$. For the other inclusion it suffices to show that, for all $a \in A$, if a finite set of names X supports $m(a)$ then it also supports a. Let $a \in A$ and let X be a finite set of names supporting $m(a)$. Let $\pi \in Aut(\mathbf{N})$ such that $\pi(n) = n$ holds for all $n \in X$. Then we have $\pi \cdot m(a) = m(a)$ since X supports $m(a)$, and we have $\pi \cdot m(a) = m(\pi \cdot a)$ by equivariance of m. This gives $m(a) = m(\pi \cdot a)$, which implies $a = \pi \cdot a$, since m is a monomorphism. This shows that X supports a, thus completing the proof.

□

Proposition 3.3.6. *For all objects A in $\mathbb{S}$ and all elements $x \in A$ and $n \in \mathbf{N}$, the following are equivalent.*

1. *$n \in \mathrm{supp}(x)$ holds.*
2. *There exists $m \in \mathbf{N} \setminus \mathrm{supp}(x)$ such that $(m\ n) \cdot x \neq x$ holds.*
3. *For all $m \in \mathbf{N} \setminus \mathrm{supp}(x)$, the inequation $(m\ n) \cdot x \neq x$ holds.*

Proof. See [33, § 9.1]. □

Although we have given $\mathbb{S}$ the name Schanuel *topos*, we have yet to prove that it is indeed a topos. To this end we give the following proposition.

Proposition 3.3.7. *The full and faithful inclusion functor $I : \mathbb{S} \hookrightarrow Aut(\mathbf{N})$-**Sets** preserves finite limits and has a right adjoint.*

$$\mathbb{S} \underset{I}{\overset{\rho}{\underset{\hookrightarrow}{\overset{\leftarrow}{\top}}}} Aut(\mathbf{N})\text{-}\mathbf{Sets}$$

Proof. The right adjoint ρ maps an object (A, μ_A) to the object $(\rho A, \mu_{\rho A})$, where the underlying set is defined by $\rho A = \{a \in A \mid a \text{ is finitely supported wrt. } \mu_A\}$ and the action $\mu_{\rho A}$ is the restriction of μ_A to ρA. The counit $\varepsilon_A \colon I\rho A \to A$ of the adjunction $I \dashv \rho$ is the inclusion function for $\rho A \subseteq A$. To see universality of ε, consider a map $g \colon IB \to A$. Since B is an object in $\mathbb{S}$, each of its elements is finitely supported. By equivariance, $\mathrm{supp}(g(b)) \subseteq \mathrm{supp}(b)$ holds for all $b \in B$. Hence, each element in the image of g is finitely supported. Therefore, g induces a (necessarily unique) map $g^\flat \colon B \to \rho A$ by $g^\flat(b) = g(b)$, for which $g = \varepsilon_A \circ Ig^\flat$ trivially holds.

For preservation of finite limits, it suffices to show that I preserves finite products and equalisers. Observe that, since I is full and faithful, the unit of the adjunction is a natural isomorphism $Id \cong \rho I$. Since, as a right adjoint, ρ preserves limits, we have an isomorphism $I(A \times B) \cong I(\rho IA \times \rho IB) \cong I\rho(IA \times IB)$, natural in A and B. To get a natural isomorphism $I(A \times B) \cong (IA \times IB)$, it therefore suffices to show that the natural transformation $\varepsilon_{IA \times IB} \colon I\rho(IA \times IB) \to (IA \times IB)$ is an isomorphism for all A and B. This follows since $IA \times IB$ has the finite support property, so that $\varepsilon_{IA \times IB}$ is in fact the identity. That the so obtained isomorphism $I(A \times B) \cong (IA \times IB)$ is indeed an isomorphism of binary products can be seen by observing that in the following commuting diagram the top row is, by triangular identity, the identity.

$$\begin{array}{ccccccc}
IA & \xrightarrow{I\eta} & I\rho IA & = & I\rho IA & \xrightarrow{\varepsilon} & IA \\
{\scriptstyle I\pi_1}\uparrow & & {\scriptstyle I\pi_1}\uparrow & & {\scriptstyle I\rho\pi_1}\uparrow & & {\scriptstyle \pi_1}\uparrow \\
I(A \times B) & \xrightarrow[I(\eta\times\eta)]{\cong} & I(\rho IA \times \rho IB) & \xrightarrow{\cong} & I\rho(IA \times IB) & \xrightarrow[\varepsilon]{\cong} & IA \times IB
\end{array}$$

Similarly, the terminal object is preserved since $\varepsilon_1 \colon I\rho 1 \to 1$ is the identity. Finally, for preservation of equalisers, it suffices to show that $\varepsilon_E \colon I\rho E \to E$ is an isomorphism for any object E that is the domain of a monomorphism of the form $m \colon E \rightarrowtail IA$. This follows since it is easily seen from Proposition 3.3.5.2 that if the codomain of a monomorphism has the finite support property then so does the domain. □

The above proposition, together with Proposition 3.1.4, shows that we have the following situation.

$$\mathbb{S} \underset{I}{\overset{\rho}{\underset{\hookrightarrow}{\overset{\leftarrow}{\top}}}} Aut(\mathbf{N})\text{-}\mathbf{Sets} \underset{F}{\overset{R}{\underset{\leftarrow}{\overset{\leftarrow}{\overset{U}{\underset{\top}{\overset{\top}{\longrightarrow}}}}}}} \mathbf{Sets} \tag{3.3}$$

By Corollaries 3.1.15 and 3.1.16, we immediately get that $\mathbb{S}$ is a topos. Moreover, as can be seen from the proofs of these two corollaries, the forgetful functor U is both monadic and comonadic and the inclusion I is comonadic. We can use this information to calculate the structure of $\mathbb{S}$.

We start with limits and colimits in $Aut(\mathbf{N})$-**Sets**. Since the forgetful functor U is both monadic and comonadic, it creates limits and colimits in $Aut(\mathbf{N})$-**Sets**. Spelling this out, limits and colimits in $Aut(\mathbf{N})$-**Sets** are calculated as in **Sets** and equipped with the canonical action on their components. Before giving the general construction of limits and colimits in $Aut(\mathbf{N})$-**Sets** we give some frequently used special cases.

Terminal Object. The object $(1,\cdot_1)$, where $\cdot_1 \colon Aut(\mathbf{N}) \times 1 \to 1$ is the terminal map, is terminal in $Aut(\mathbf{N})$-**Sets**.

Products. The product $(A,\cdot_A) \times (B,\cdot_B)$ in $Aut(\mathbf{N})$-**Sets** is $(A \times B, \cdot_{A\times B})$, where $A \times B$ is the product in **Sets** and $\pi \cdot_{A\times B} \langle a,b\rangle = \langle \pi \cdot_A a, \pi \cdot_B b\rangle$.

Pullbacks. The pullback of two morphisms $f \colon (A,\cdot_A) \to (C,\cdot_C)$ and $g \colon (B,\cdot_B) \to (C,\cdot_C)$ is given by

$$\begin{array}{ccc} (A \times_C B, \cdot_{A\times_C B}) & \xrightarrow{\pi_2} & (B,\cdot_B) \\ {\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle g} \\ (A,\cdot_A) & \xrightarrow[f]{} & (C,\cdot_C) \end{array}$$

where $A \times_C B = \{\langle a,b\rangle \in A \times B \mid f(a) = g(b)\}$ and $\pi \cdot_{A\times_C B} \langle a,b\rangle = \langle \pi \cdot_A a, \pi \cdot_B b\rangle$ and π_1 and π_2 are the evident projections.

Initial Object. The object $(0,\cdot_0)$, where $\cdot_0 \colon Aut(\mathbf{N}) \times 0 \to 0$ is the unique map of its type, is initial in $Aut(\mathbf{N})$-**Sets**.

Coproducts. The coproduct $(A,\cdot_A) + (B,\cdot_B)$ in $Aut(\mathbf{N})$-**Sets** is $(A+B,\cdot_{A+B})$, where $A+B$ is the coproduct in **Sets** and $\pi \cdot_{A+B} \kappa_1(a) = \kappa_1(\pi \cdot_A a)$ and $\pi \cdot_{A+B} \kappa_2(b) = \kappa_2(\pi \cdot_B b)$.

Coequalisers. The coequaliser of two morphisms $f,g \colon (A,\cdot_A) \to (B,\cdot_B)$ is given by

$$(A,\cdot_A) \underset{g}{\overset{f}{\rightrightarrows}} (B,\cdot_B) \xrightarrow{\;e\;} (E,\cdot_E),$$

in which E and e are defined by $E \stackrel{\text{def}}{=} B/_\sim$ and $e(b) \stackrel{\text{def}}{=} [b]_\sim$ for the equivalence relation $\sim$ generated by $b \sim b' \iff \exists a \in A.\, f(a) = b \wedge g(a) = b'$. The action $\cdot_E$ is given by $\pi \cdot_E [b]_\sim = [\pi \cdot_B b]$.

In general, for a diagram $J \colon \mathbb{J} \to Aut(\mathbf{N})$-**Sets**, limits and colimits can be defined as follows.

$$\begin{array}{rcl} \lim J & = & \{p \in \Pi_{j\in Obj(\mathbb{J})} J(j) \mid \forall (f \colon j \to k) \in Mor(\mathbb{J}).\ p_k = J(f)(p_j))\} \\ \pi \cdot_{\lim J} p & = & \lambda j.\ \pi \cdot_{J(j)} p_j \\ \operatorname{colim} J & = & (\Sigma_{j\in Obj(\mathbb{J})} J(j))/_\sim \\ & & \text{where } \langle j,x\rangle \sim \langle k,y\rangle \iff \exists (f \colon j \to k) \in Mor(\mathbb{J}).\ y = J(f)(x) \\ \pi \cdot_{\operatorname{colim} J} [\langle j,x\rangle]_\sim & = & [\langle j, \pi \cdot_{J(j)} x\rangle]_\sim \end{array}$$

We now consider limits in $\mathbb{S}$. Let $J \colon \mathbb{J} \to \mathbb{S}$ be a diagram of shape $\mathbb{J}$ in $\mathbb{S}$. Since I is full and faithful, the unit of the adjunction $I \vdash \rho$ is a natural isomorphism $Id \cong \rho I$. Since ρ is right adjoint, it preserves limits, so that we have a natural isomorphism $\lim J \cong \lim \rho I J \cong \rho(\lim IJ)$. This shows that $\mathbb{S}$ has all limits which exist in $Aut(\mathbf{N})$-**Sets** and that limits in $\mathbb{S}$ are taken by first taking the limit in $Aut(\mathbf{N})$-**Sets** and then restricting this limit to all the elements with finite support. In the proof of Proposition 3.3.7,

we have seen that $I\rho$ acts as the identity functor on finite limits, which means that finite limits in $\mathbb{S}$ can be taken exactly as in $Aut(\mathbf{N})$-**Sets**.

For colimits in $\mathbb{S}$, we observe that the inclusion I, as a comonadic functor, creates colimits. Therefore, $\mathbb{S}$ has all colimits which exist in $Aut(\mathbf{N})$-**Sets**. Moreover, since I is just the inclusion functor, colimits in $\mathbb{S}$ can be defined exactly as in $Aut(\mathbf{N})$-**Sets**. For a diagram $J\colon \mathbb{J} \to \mathbb{S}$, the colimit $\operatorname{colim} IJ$ in $Aut(\mathbf{N})$-**Sets** has the finite support property, i.e. is an object in $\mathbb{S}$, and is colimiting for J.

We come to the cartesian closed structure. In $Aut(\mathbf{N})$-**Sets** the exponent $(A,\cdot_A) \Rightarrow (B,\cdot_B)$ is given by $(A \Rightarrow B, \cdot_{A\Rightarrow B})$, where $A \Rightarrow B$ is the exponent in **Sets** and $\cdot_{A\Rightarrow B}$ is the equivariant action $\pi \cdot_{A\Rightarrow B} f = \lambda a.\ \pi \cdot_B (f(\pi^{-1} \cdot_A a))$. Note that the underlying set $A \Rightarrow B$ contains all functions from A to B, not just equivariant ones. To define the cartesian closed structure in $\mathbb{S}$, observe that, for any map $C \times A \to B$ in $\mathbb{S}$, we have the following natural one-to-one correspondences.

$$\dfrac{\dfrac{\dfrac{\dfrac{C \times A \longrightarrow B}{I(C \times A) \longrightarrow IB}\ I \text{ is full and faithful}}{IC \times IA \longrightarrow IB}\ I \text{ preserves finite limits}}{IC \longrightarrow (IA \Rightarrow IB)}\ \text{exponential transpose}}{C \longrightarrow \rho(IA \Rightarrow IB)}\ I \text{ is left adjoint to } \rho$$

The functor $\rho(IA \Rightarrow I-)$ is therefore right adjoint to $- \times A$, so that the exponent $(A \Rightarrow B)$ in $\mathbb{S}$ can be defined as $\rho(IA \Rightarrow IB)$. It consists of all the functions in $IA \Rightarrow IB$ that have finite support.

A subobject classifier $true\colon 1 \to \Omega$ in both $Aut(\mathbf{N})$-**Sets** and $\mathbb{S}$ is given by the map $\kappa_1\colon 1 \to 1+1$. The unique classifying map $\chi_m\colon A \to \Omega$ of a monomorphism $m\colon B \rightarrowtail A$ as in

$$\begin{array}{ccc} B & \longrightarrow & 1 \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle true} \\ A & \xrightarrow[\chi_m]{} & \Omega \end{array}$$

is given by $\chi(a) = \kappa_1$ if $(\exists b \in B.\, a = m(b))$ and $\chi(a) = \kappa_2$ otherwise. The power object $\mathscr{P}(A)$ is as usual defined as $(A \Rightarrow \Omega)$. In $Aut(\mathbf{N})$-**Sets**, the power object may be described as having underlying set $\{A' \mid A' \subseteq A \text{ in } \mathbf{Sets}\}$ and action $\pi \cdot_{\mathscr{P}(A)} A' = \{\pi \cdot_A a \mid a \in A'\}$. In $\mathbb{S}$, the above construction of the cartesian closed structure implies that the power object $\mathscr{P}(A)$ may be described as having underlying set $\{A' \mid A' \subseteq A \text{ in } \mathbf{Sets} \text{ and } \operatorname{supp}(A') \text{ is finite}\}$ with action defined as in $Aut(\mathbf{N})$-**Sets**.

Partial map classifiers can be defined from the topos structure of $\mathbb{S}$. For an object A of $\mathbb{S}$, the partial map classifier $\eta_A\colon A \to A_\perp$ with the property that, for each monomorphism $m\colon B \rightarrowtail C$ and each map $f\colon B \to A$, there exists a unique classifying map $\chi\colon C \to A_\perp$ making the diagram

$$\begin{array}{ccc} B & \xrightarrow{f} & A \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta_A} \\ C & \xrightarrow[\chi]{} & A_\perp \end{array}$$

a pullback, may be defined as $A_\perp = A + 1$ and $\eta_A(a) = \kappa_1(a)$. We write $\perp$ for the element $\kappa_2(\diamond) \in A_\perp$.

Remark 3.3.8. The above construction of $\mathbb{S}$ is an instance of a more general situation. Let G be a topological group, i.e. G is a group object in the category **Top** of topological spaces. For a set A write A^δ for the set A equipped with the discrete topology. Let $\mathbf{B}(G)$ be the full subcategory of G-**Sets** consisting of all the objects (A, μ_A) for which the action $\mu_A \colon G \times A^\delta \to A^\delta$ in continuous, i.e. μ_A is a morphism in **Top**. The category G-**Sets** appears as $\mathbf{B}(G^\delta)$. With these definitions, we have the situation

$$\mathbf{B}(G) \underset{I}{\overset{\rho}{\underset{\hookrightarrow}{\leftarrow}}} \top \; \mathbf{B}(G^\delta) \underset{F}{\overset{R}{\underset{\top}{\overset{\top}{\rightleftarrows}}}} \mathbf{Sets},$$

where U is the forgetful functor and I is the full, faithful and finite limit preserving inclusion. See [64, §VII.3] for details on this situation.

The Schanuel topos $\mathbb{S}$ arises as $\mathbf{B}(G)$ when G is taken to be the group $Aut(\mathbf{N})$ equipped with the topology inherited from the product topology on $\mathbf{N} \Rightarrow \mathbf{N}$. Details appear in [64, §III.9] and [39]. □

Remark 3.3.9. In the definition of $\mathbb{S}$, we could have used the group $Aut_{\mathrm{fk}}(\mathbf{N})$ of permutations with finite support instead of the group $Aut(\mathbf{N})$ of all automorphisms on $\mathbf{N}$. A permutation $\pi \in Aut(\mathbf{N})$ has finite support if it satisfies

$$\exists X \in \mathscr{P}_{\mathrm{fin}}(\mathbf{N}).\ \forall n \in (\mathbf{N} \setminus X).\ \pi(n) = n.$$

That finite support permutations are enough is a consequence of the finite support property. For an object $(A, \cdot_A)$ of $\mathbb{S}$ and an element $a \in A$, the action of a permutation π on a is completely determined by the values of π on the support of a:

$$\forall \pi, \pi' \in Aut(\mathbf{N}).\ (\forall n \in \mathrm{supp}(a).\ \pi(n) = \pi'(n)) \implies (\pi \cdot_A a = \pi' \cdot_A a)$$

Since $\mathrm{supp}(a)$ is a finite set, the action on a of arbitrary permutations can therefore be reconstructed from the action on a of finite support permutations. Details of the construction of $\mathbb{S}$ using finite support permutations and a proof of the equivalence to the above presentation appear in [39].

It is a standard result of group theory that the group $Aut_{\mathrm{fk}}(\mathbf{N})$ is generated by all the swappings $(n\, n')$, where, for n and n' in $\mathbf{N}$, $(n\, n')$ is the permutation defined by

$$(n\, n')(m) = \begin{cases} n' & \text{if } m = n \\ n & \text{if } m = n' \\ m & \text{otherwise.} \end{cases}$$

Finally, we note that the group $Aut_{\mathrm{fk}}(\mathbf{N})$ becomes an object of $\mathbb{S}$ when it is equipped with the action $\pi \cdot_{Aut_{\mathrm{fk}}(\mathbf{N})} \tau = \pi \circ \tau \circ \pi^{-1}$, i.e. the action of the exponent $\mathbf{N} \Rightarrow \mathbf{N}$. This does not hold for the full group $Aut(\mathbf{N})$, as it does not have the finite support property. □

3.3.1 Monoidal Closed Structure and Simple Monoidal Products

In this section we consider freshness and show that it gives rise to a strict affine symmetric monoidal closed structure.

Definition 3.3.10. Let A and B be objects of $\mathbb{S}$ and let $a \in A$ and $b \in B$. We say that *a is fresh for b*, written as $a \# b$, if $\mathrm{supp}(a) \cap \mathrm{supp}(b) = \emptyset$ holds.

Proposition 3.3.11. *For all objects A, B and C in $\mathbb{S}$, and all elements $a \in A$, $b \in B$ and $c \in C$, the following are true.*

1. *$a \# b$ implies $(\pi \cdot a) \# (\pi \cdot b)$ for all permutations π.*
2. *$a \# \diamond$, where $\diamond$ is the unique element of 1.*
3. *$a \# b$ implies $b \# a$.*
4. *$a \# b \wedge \langle a, b\rangle \# c$ if and only if $a \# \langle b, c\rangle \wedge b \# c$.*

Proof. Immediate from the definition of #. □

By this proposition, the definitions

$$A * B \stackrel{\text{def}}{=} \{\langle a, b\rangle \in A \times B \mid a \# b\} \qquad \pi \cdot_{A*B} \langle a, b\rangle \stackrel{\text{def}}{=} \langle \pi \cdot_A a, \pi \cdot_B b\rangle$$

define a strict affine symmetric monoidal structure $* \colon \mathbb{S} \times \mathbb{S} \to \mathbb{S}$.

Proposition 3.3.12. *For each object A in $\mathbb{S}$, the functor $- * A \colon \mathbb{S} \to \mathbb{S}$ preserves pullbacks.*

Proof. This follows from the fact that $\langle a, b\rangle \# c$ holds if and only if both $a \# c$ and $b \# c$ hold. □

We use the construction from Section 3.2 to show that this monoidal structure is closed.

Lemma 3.3.13. *For each object A in $\mathbb{S}$, the functor $- * A \colon \mathbb{S} \to \mathbb{S}$ preserves exact sequences.*

Proof. Let

$$R \underset{r_2}{\overset{r_1}{\rightrightarrows}} B \xrightarrow[c]{} C$$

be an exact sequence, i.e. $\langle r_1, r_2\rangle$ is the kernel pair of c, and c is the coequaliser of r_1 and r_2. As any kernel pair, $\langle r_1, r_2\rangle$ is an equivalence relation. By the construction of pullbacks, we can assume $R \subseteq B \times B$ and $r_1 = \pi_1$ and $r_2 = \pi_2$. We write bRb' if b and b' are related by R and $[b]_R$ for the equivalence class of b under R. By the construction of coequalisers, we can assume both $C = B/R$ and $c(b) = [b]_R$.

We have to show that

$$R * A \underset{r_2 * A}{\overset{r_1 * A}{\rightrightarrows}} B * A \xrightarrow[c * A]{} C * A$$

is an exact sequence. In this diagram, $R * A \subseteq (B * A) \times (B * A)$ is the equivalence relation defined such that $\langle b, a\rangle (R * A) \langle b', a'\rangle$ holds if and only if both $a = a'$ and bRb' hold. That $(r_1 * A, r_2 * A)$ is a kernel pair of $c * A$ follows directly from pullback-preservation of $(-) * A$. It remains to show that $c * A$ is

a coequaliser of $r_1 * A$ and $r_2 * A$. We have to define an isomorphism $i\colon (B/R) * A \to (B * A)/(R * A)$ satisfying $i(\langle [b]_R, a\rangle) = [b,a]_{R*A}$ for all $b \in B$ and $a \in A$ for which $b \# a$ holds.

To define i, let $\langle [b]_R, a\rangle$ be an element of $(B/R) * A$. Let π be a permutation that exchanges all names in the support of a for fresh names and leaves all other names unchanged. Because of $[b]_R \# a$, we have $\pi \cdot [b]_R = [b]_R$. Since the permutation acts pointwise on equivalence classes, $[\pi \cdot b]_R = \pi \cdot [b]_R = [b]_R$ holds. By definition of π, we also have $(\pi \cdot b) \# a$. Define $i(\langle [b]_R, a\rangle) = [\pi \cdot b, a]_{R*A}$. The definition of $R * A$ makes it evident that this definition is independent of the choice of b and π.

It just remains to show that i is an isomorphism. Its inverse i^{-1} is defined by $i^{-1}([b,a]_{R*A}) = \langle [b]_R, a\rangle$. Note that $[b]_R \# a$ holds since the pointwise definition of the action on $[b]_R$ implies $\mathrm{supp}([b]_R) \subseteq \mathrm{supp}(b)$. Showing that i and i^{-1} are indeed inverses is straightforward. □

Proposition 3.3.14. *For any object A of $\mathbb{S}$, the functor $- * A\colon \mathbb{S} \to \mathbb{S}$ has a right adjoint $A \multimap -\colon \mathbb{S} \to \mathbb{S}$.*

Proof. We use Proposition 3.2.5, for the application of which we need to verify properties (C1), (C2) and (C3) for $\mathbb{S}$. Property (C1) holds since in a topos any monomorphism strong. Property (C3) is shown in Lemma 3.3.13. It remains to show property (C2). For $f\colon B * A \to C$, the map $\tilde{f}\colon B \times A \to C_\perp$ is given by

$$\tilde{f}(b,a) = \begin{cases} f(b,a) & \text{if } b \# a \text{ holds} \\ \perp & \text{otherwise.} \end{cases}$$

Furthermore, the map $(\Lambda\tilde{f})\colon B \to (A \Rightarrow C_\perp)$ is defined by $(\Lambda\tilde{f})(b) = \lambda c.\, \tilde{f}(b,c)$. For (C2) it suffices to show that, for all $b \in B$ and all $a \in A$, we have $b \# a$ if and only if $(\Lambda\tilde{f})(b) \# a$. The direction from left to right follows by equivariance of $(\Lambda\tilde{f})$. In the other direction, suppose for a contradiction that we have $b \in B$ and $a \in A$ with $(\Lambda\tilde{f})(b) \# a$ and $\neg(b \# a)$. Since the set of names is infinite and $\mathrm{supp}(a)$ is finite, we can always find a permutation π that swaps all the names in $\mathrm{supp}(a)$ with fresh names and fixes all the names in $\mathrm{supp}(b) \setminus \mathrm{supp}(a)$. Then we have $b \# \pi \cdot a$. Since $(\Lambda\tilde{f})(b) \# a$, by assumption, and $\mathrm{supp}((\Lambda\tilde{f})(b)) \subseteq \mathrm{supp}(b)$, by equivariance, we have $\mathrm{supp}((\Lambda\tilde{f})(b)) \subseteq \mathrm{supp}(b) \setminus \mathrm{supp}(a)$. Since π fixes all names in $\mathrm{supp}(b) \setminus \mathrm{supp}(a)$, this implies $\pi \cdot ((\Lambda\tilde{f})(b)) = (\Lambda\tilde{f})(b)$. Then we have

$$\begin{array}{rcll} \perp & = & \tilde{f}(b,a) & \text{by definition, since } \neg(b \# a) \\ & = & (\Lambda\tilde{f})(b)(a) & \text{by definition} \\ & = & (\pi \cdot (\Lambda\tilde{f})(b))(a) & \text{since } (\Lambda\tilde{f})(b) = \pi \cdot ((\Lambda\tilde{f})(b)) \\ & = & (\Lambda\tilde{f})(\pi \cdot b)(a) & \text{by equivariance of } \Lambda\tilde{f} \\ & = & \tilde{f}(\pi \cdot b, a) & \text{by definition} \\ & = & f(\pi \cdot b, a) \in C & \text{by definition, since } \pi \cdot b \# a \end{array}$$

Since $\perp \in C_\perp$ is, by definition, different from any element $c \in C$, we have therefore arrived at a contradiction, so that our assumption $(\Lambda\tilde{f})(b) \# a$ and $\neg(b \# a)$ must have been false. Hence, property (C2) holds. □

By Proposition 2.5.6, the monoidal closed structure $(*, \multimap)$ provides us with simple monoidal products Π^* in the codomain fibration $cod\colon \mathbb{S}^{\to} \to \mathbb{S}$.

Expanding the construction in Proposition 3.2.5, the object $A \multimap B$ is defined as follows.

$$\begin{aligned} E &\stackrel{\text{def}}{=} \{f \in (A \Rightarrow B_\perp) \mid \forall a \in A.\ a \# f \iff f(a) \neq \perp\} \\ \sim &\stackrel{\text{def}}{=} \{\langle f,g\rangle \in E \times E \mid \forall a \in A.\ a \# \langle f,g\rangle \implies f(a) = g(a)\} \\ (A \multimap B) &\stackrel{\text{def}}{=} E/\sim \end{aligned}$$

Here, $\sim$ is a simplification of the equivalence relation R used in the construction of Proposition 3.2.5. The equivalence relation there amounts to:

$$R \stackrel{\text{def}}{=} \{\langle f,g\rangle \in E \times E \mid \exists s \in (A \Rightarrow A_\perp).\ \forall a \in A.\ a \# \langle f,g,s\rangle \implies f(a) = g(a)\}$$

However, we have $\sim = R$. The inclusion $\sim \subseteq R$ is trivial. For $R \subseteq \sim$, assume fRg and $a \in A$ with $a \# \langle f,g\rangle$. From the assumption fRg we obtain $s \in (A \Rightarrow A_\perp)$ such that $\forall a \in A.\ a \# \langle f,g,s\rangle \implies f(a) = g(a)$ holds. Let π be a permutation swapping the names in a for fresh names and leaving all other names unchanged. Then we get $\pi \cdot f = f$ and $\pi \cdot g = g$, which implies $\pi \cdot (f(a)) = f(\pi \cdot a)$ and $\pi \cdot (g(a)) = g(\pi \cdot a)$. Since $\pi \cdot a$ is fresh, in particular for $\langle f,g,s\rangle$, we get $f(\pi \cdot a) = g(\pi \cdot a)$ from the assumption. Hence, we have $\pi \cdot (f(a)) = \pi \cdot (g(a))$, which implies $f(a) = g(a)$, thus showing $f \sim g$.

Note that, by construction, R is a subobject of $E \times E$, which implies that both R and $\sim$ are equivariant equivalence relations.

The elements of $(A \multimap B)$, which are by definition equivalence classes, can also be described as partial functions from A to B. By this we mean that $(A \multimap B)$ is a subobject of $(A \Rightarrow B_\perp)$, as shown in the following proposition.

Proposition 3.3.15. *For all objects A and B in $\mathbb{S}$, the following assignment defines a monomorphism $m\colon (A \multimap B) \rightarrowtail (A \Rightarrow B_\perp)$ in $\mathbb{S}$.*

$$m(c) \stackrel{\text{def}}{=} \lambda x\colon A.\ \begin{cases} f(x) & \text{if } f \in c \text{ and } f(x) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

Moreover, the diagram below commutes.

$$\begin{array}{ccccc} (A \multimap B) * A & \xrightarrow{m \times A} & (A \Rightarrow B_\perp) * A & \xrightarrow{\iota} & (A \Rightarrow B_\perp) \times A \\ \downarrow{\scriptstyle \varepsilon} & & & & \downarrow{\scriptstyle \mathrm{ev}} \\ B & & \xrightarrow[\eta_B]{} & & B_\perp \end{array}$$

Proof. We start be showing that m is well-defined, i.e. that it is a morphism from $(A \multimap B)$ to $(A \Rightarrow B_\perp)$ in $\mathbb{S}$. To this end we first show that, for all $c \in (A \multimap B)$, $m(c)$ is a partial function from A to B. We have to show that, for all $f,g \in c$, if $f(x) \neq \perp$ and $g(x) \neq \perp$ hold then so does $f(x) = g(x)$. Suppose $f,g \in c$ and $f(x) \neq \perp$ and $g(x) \neq \perp$. Since $f,g \in E$, this implies $x \# f$ and $x \# g$, which further implies $x \# \langle f,g\rangle$. Since $f,g \in c$, we have $f \sim g$, which by definition of $\sim$ implies the required $f(x) = g(x)$.

Next, it is straightforward to verify that m is equivariant. As a consequence of equivariance, the support of $m(c)$ is contained in the support of c. Therefore, $m(c)$ is a partial function from A to B with

finite support, i.e. an element of $(A \Rightarrow B_\perp)$. Putting this together we have shown that m is a morphism in $\mathbb{S}$ from $(A \multimap B)$ to $(A \Rightarrow B_\perp)$.

That m is a morphism in $\mathbb{S}$ can also be seen because it can be defined in the internal language as partial function with $\mathrm{graph}(m(c)) = \bigcup_{f \in c} \mathrm{graph}(f)$. The above argument that $\mathrm{graph}(m(c))$ is the graph of a partial function can be carried out internally.

Next we show that m is a monomorphism, i.e. if $m(c) = m(d)$ holds then so does $c = d$. Suppose $m(c) = m(d)$. Let f and g be arbitrary elements of the equivalence classes c and d respectively. By definition of m and $m(c) = m(d)$, we obtain that $f(x) = g(x)$ holds for all $x \in A$ for which both $f(x) \neq \perp$ and $g(x) \neq \perp$ hold. Since both f and g are in E, this implies $f(x) = g(x)$ for all $x \in A$ with $x \# \langle f, g \rangle$. By definition of $\sim$, this implies $f \sim g$. But then the equivalence classes c and d are not disjoint and so must be equal. This shows that m is a monomorphism.

Finally, it is straightforward to see that the diagram commutes. □

Next we show that, for an equivalence class c in $(A \multimap B)$, the partial function $m(c)$ not only represents the equivalence class c but is in fact a canonical element of it. For this we use three sub-lemmas.

Lemma 3.3.16. *For all objects A in $\mathbb{S}$ and all elements $c \in \mathscr{P}(A)$, we have $\bigcap_{x \in c} \mathrm{supp}(x) \subseteq \mathrm{supp}(c)$*

Proof. Suppose, for a contradiction, that there exists an $n \in \mathbf{N}$ with $n \in \bigcap_{x \in c} \mathrm{supp}(x)$ but not $n \in \mathrm{supp}(c)$. This means that $\forall x \in c.\ n \in \mathrm{supp}(x)$ and $n \notin \mathrm{supp}(c)$ hold. Let m be a fresh name. By Proposition 3.3.6, we have $(m\ n) \cdot c = c$. By definition, we have $(m\ n) \cdot c = \{(m\ n) \cdot x \mid x \in c\}$. Now, if $n \in \mathrm{supp}(x)$ and $m \neq n$ then $n \notin \mathrm{supp}((m\ n) \cdot x)$. By definition of $(m\ n) \cdot c$ and because we have assumed $n \in \mathrm{supp}(x)$ for all $x \in c$, this implies $n \notin \mathrm{supp}(y)$ for all $y \in (m\ n) \cdot c$. But by assumption $n \in \mathrm{supp}(x)$ holds for all $x \in c$. Therefore, c and $(m\ n) \cdot c$ cannot contain the same elements, giving $(m\ n) \cdot c \neq c$ and thus the required contradiction. □

Lemma 3.3.17. *Let A be an object in $\mathbb{S}$ and $\sim$ be an equivariant equivalence relation on A. For all equivalence classes $c \in (A/\sim)$, we have* $\mathrm{supp}(c) \subseteq \bigcap_{x \in c} \mathrm{supp}(x)$.

Proof. Equivariant equivalence relations in $\mathbb{S}$ amount to equivalence relations in the topos-theoretic sense. Since in a topos quotients appear as coequalisers, there exists a map $A \to (A/\sim)$ in $\mathbb{S}$ mapping x to $[x]_\sim$. By equivariance, this implies $\mathrm{supp}([x]_\sim) \subseteq \mathrm{supp}(x)$ for all $x \in A$. Therefore we have $\mathrm{supp}(c) \subseteq \mathrm{supp}(x)$ for all $x \in c$. But this implies the required inclusion $\mathrm{supp}(c) \subseteq \bigcap_{x \in c} \mathrm{supp}(x)$. □

Lemma 3.3.18. *Let A and B be objects of $\mathbb{S}$ and let $m\colon (A \multimap B) \rightarrowtail (A \Rightarrow B_\perp)$ be the monomorphism from Proposition 3.3.15. For each equivalence class c in $(A \multimap B)$, we have* $\mathrm{supp}(m(c)) = \bigcap_{f \in c} \mathrm{supp}(f)$.

Proof. We have $\mathrm{supp}(c) = \bigcap_{f \in c} \mathrm{supp}(f)$ by Lemmas 3.3.16 and 3.3.17 and $\mathrm{supp}(m(c)) = \mathrm{supp}(c)$ by Proposition 3.3.5. □

Lemma 3.3.19. *Let A and B be objects in $\mathbb{S}$ and let $m\colon (A \multimap B) \rightarrowtail (A \Rightarrow B_\perp)$ be the monomorphism from Proposition 3.3.15. For each equivalence class c in $(A \multimap B)$, we have $m(c) \in c$.*

Proof. We start by showing $m(c) \in E$, i.e. that $x \# m(c) \iff m(c)(x) \neq \bot$ holds for all $x \in A$. Let $x \in A$. From left to right, assume $x \# m(c)$. Let f be an arbitrary element of c. Suppose for a contradiction that $m(c)(x) = \bot$ holds. Let π be a permutation that exchanges all the names in the support of x with fresh names and that leaves all other names fixed. Since $x \# m(c)$, this implies $\pi \cdot (m(c)) = m(c)$. Moreover, $\pi \cdot x$ is fresh for f and c. Since f is in E, this implies $f(\pi \cdot x) \neq \bot$. Therefore, we have

$$\begin{aligned}
\bot \neq f(\pi \cdot x) \quad & \text{since } f \in E \text{ and } \pi \cdot x \# f \\
= m(c)(\pi \cdot x) \quad & \text{by definition of } m(c) \\
= \pi \cdot (m(c)(x)) \quad & \text{since } \pi \cdot (m(c)) = m(c), \text{ which follows from } x \# m(c) \text{ by def. of } \pi \\
= \bot \quad & \text{by assumption } m(c)(x) = \bot,
\end{aligned}$$

which gives the required contradiction.

From right to left, assume $m(c)(x) \neq \bot$. By definition of $m(c)$, there exists $f \in c$ such that $f(x) \neq \bot$. Since f is in E, this implies $x \# f$. Since $\mathrm{supp}(m(c)) = \bigcap_{f \in c} \mathrm{supp}(f)$ holds by the previous lemma, this implies $x \# m(c)$, as required.

Finally, $m(c) \in c$ follows since it is easily shown that $m(c) \sim f$ holds for all $f \in c$. □

By Proposition 3.3.15 and the previous lemmas we can now characterise the partial functions in the image of $m \colon (A \multimap B) \rightarrowtail (A \Rightarrow B_\bot)$ as the maximally defined functions among all partial functions that are defined just on fresh arguments. The following proposition makes this precise.

Proposition 3.3.20. *For all objects A and B, the set $(A \multimap B)$ has the following characterisation.*

$$\begin{aligned}
(A \multimap B) &\cong \{f \in E \mid \forall g \in E.\ f \sim g \implies dom(f) \supseteq dom(g)\} \\
&\cong \{f \in E \mid \forall g \in E.\ f \sim g \implies \mathrm{supp}(f) \subseteq \mathrm{supp}(g)\}
\end{aligned}$$

Proof. First we show $(A \multimap B) \cong \{f \in E \mid \forall g \in E.\ f \sim g \implies dom(f) \supseteq dom(g)\}$. By Proposition 3.3.15 and Lemma 3.3.19, each equivalence class $c \in (A \multimap B)$ has a canonical element $m(c)$. By definition of m and the equivalence relation $\sim$, the element $m(c)$ must be maximally defined, i.e. satisfy $dom(m(c)) \supseteq dom(h)$ for all $h \in c$. Moreover, the equivalence class c can contain at most one such maximally defined element. Suppose f and g were two elements of c satisfying $dom(f) \supseteq dom(h)$ and $dom(g) \supseteq dom(h)$ for all $h \in c$. By letting h be f and g, this implies $dom(f) = dom(g)$. Since both f and g are in E and because of $dom(f) = dom(g)$, we get $x \# f \iff f(x) \neq \bot \iff g(x) \neq \bot \iff x \# g$ for all $x \in A$. By $f \sim g$, this implies $f(x) = g(x)$ for all $x \in dom(f) = dom(g)$. But for $x \notin dom(f) = dom(g)$, we trivially have $\bot = f(x) = g(x)$, so that we have shown $f = g$, as required for uniqueness.

To show $(A \multimap B) \cong \{f \in E \mid \forall g \in E.\ f \sim g \implies \mathrm{supp}(f) \subseteq \mathrm{supp}(g)\}$ it suffices to show that, for each $c \in (A \multimap B)$, $m(c)$ is the element of c with the least support. Lemma 3.3.18 shows that $m(c)$ has minimal support of all the elements of c. By definition of $\sim$, there can be only one such element of c, thus showing the claimed isomorphism. □

For name-like objects A, a simpler characterisation of $(A \multimap B)$ is available, see Proposition 10.3.11.

Remark 3.3.21. The Schanuel topos $\mathbb{S}$ is equivalent to the category $\mathbf{Sets}^{\mathbb{I}}_{pbp}$ of pullback-preserving functors in $\mathbf{Sets}^{\mathbb{I}}$, where $\mathbb{I}$ is the category of finite sets and injections, see e.g. [64, §III.9]. Using this equivalence, the monoidal closed structure $(*, -\!\!*)$ on $\mathbb{S}$ can (up to isomorphism) be defined by lifting the disjoint union monoidal structure on $\mathbb{I}$ using Day's construction, see e.g. [4]. Indeed, as a sheaf, the object $(A -\!\!* B)$ has the simple description $(A -\!\!* B)(S) \cong \mathbf{Sets}^{\mathbb{I}}_{pbp}(A(-), B(S+-))$, see [103]. It is not clear whether or not this presentation can be used to simplify the above construction. The main reason for preferring the above direct construction of $-\!\!*$ is that it also works for assemblies in the next section, for which Day's construction is not available. □

3.3.2 Simple Monoidal Sums

We now come to the construction of simple monoidal sums Σ^*. It is instructive to consider first the definition of $\Sigma^*_{\mathbf{N}}$ for the object of names $\mathbf{N}$. In Section 2.5.1 we have defined simple monoidal sums $\Sigma^*_{\mathbf{N}}$ as a fibred functor $\Sigma^*_{\mathbf{N}} \colon Gl(- * \mathbf{N}) \to cod$ that is a fibred left adjoint to $W_{\mathbf{N}} \colon cod \to Gl(- * \mathbf{N})$.

$$\begin{array}{ccc} \mathbb{S}/(- * \mathbf{N}) & \xrightarrow{\Sigma^*_{\mathbf{N}}} & \mathbb{S}^{\to} \\ & {\scriptstyle Gl(-*\mathbf{N})} \searrow \quad \swarrow {\scriptstyle cod} & \\ & \mathbb{S} & \end{array} \tag{3.4}$$

As explained informally in the introduction, $\Sigma^*_{\mathbf{N}}$ generalises the abstraction set of Gabbay & Pitts [38]. Just as the abstraction set, $\Sigma^*_{\mathbf{N}}$ is constructed as a certain quotient. Let $f \colon B \to \Gamma * \mathbf{N}$ be an object of $\mathbb{S}/(- * \mathbf{N})$. Write f_1 and f_2 as abbreviations for the maps $\dot{\pi}_1 \circ f \colon B \to \Gamma$ and $\dot{\pi}_2 \circ f \colon B \to \mathbf{N}$ respectively. The equivalence relation $\sim_f \subseteq B \times B$ with respect to which the quotient is taken is defined by

$$b \sim_f b' \iff \big(\exists n \in \mathbf{N}.\, n \# \langle b, b' \rangle \wedge (n\ f_2(b)) \cdot b = (n\ f_2(b')) \cdot b'\big). \tag{3.5}$$

By the some/any property from [38], we also have

$$b \sim_f b' \iff \big(\forall n \in \mathbf{N}.\, n \# \langle b, b' \rangle \implies (n\ f_2(b)) \cdot b = (n\ f_2(b')) \cdot b'\big). \tag{3.6}$$

With this it is straightforward to see that $\sim_f$ an equivariant equivalence relation. We write $[b]_f$ for the equivalence class of b with respect to this equivalence relation. Mirroring Lemma 5.1 of [38], the equivalence class $[b]_f$ can be described as

$$[b]_f = \{(n\ f_2(b)) \cdot b \mid n \in \mathbf{N} \wedge (n = f_2(b) \vee n \# f_2(b))\}. \tag{3.7}$$

We define the functor $\Sigma^*_{\mathbf{N}}$ directly. It maps an object $f \colon B \to \Gamma * \mathbf{N}$ to the object $\Sigma^*_{\mathbf{N}} f \colon \Sigma^*_{\mathbf{N}} B \to \Gamma$ given by $\Sigma^*_{\mathbf{N}} B = \{[b]_f \mid b \in B\}$ with action $\pi \cdot_{\Sigma^*_{\mathbf{N}} B} [b]_f = [\pi \cdot_B b]_f$ and $(\Sigma^*_{\mathbf{N}} f)([b]_f) = f_1(b)$. A morphism $(u, v) \colon f \to g$ in $\mathbb{S}/(- * \mathbf{N})$, as given by

$$\begin{array}{ccc} B & \xrightarrow{u} & C \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ \Gamma * \mathbf{N} & \xrightarrow[v*\mathbf{N}]{} & \Delta * \mathbf{N} \end{array} \tag{3.8}$$

is mapped by $\Sigma^*_{\mathbf{N}}$ to the morphism

$$\begin{array}{ccc} \Sigma^*_{\mathbf{N}}B & \xrightarrow{\Sigma^*_{\mathbf{N}}u} & \Sigma^*_{\mathbf{N}}C \\ {\scriptstyle \Sigma^*_{\mathbf{N}}f}\downarrow & & \downarrow{\scriptstyle \Sigma^*_{\mathbf{N}}g} \\ \Gamma & \xrightarrow[v]{} & \Delta \end{array} \tag{3.9}$$

defined by $(\Sigma^*_{\mathbf{N}}u)([b]_f) = [u(b)]_g$. We need to verify that $\Sigma^*_{\mathbf{N}}$ is well-defined. For the object part, we have to check that the definition $(\Sigma_{\mathbf{N}}f)([b]_f) = f_1(b)$ does not depend on the choice of representative b, i.e. we have to show $f_1(b) = f_1(b')$ for all b and b' in B that satisfy $[b]_f = [b']_f$. To this end it suffices to show that $b \sim_f b'$ implies $f_1(b) = f_1(b')$. If $b \sim_f b'$ holds then there exists a name $n \in \mathbf{N}$, fresh for b and b', such that $(n\ f_2(b)) \cdot b = (n\ f_2(b')) \cdot b'$ holds. This equality and equivariance of f_1 imply

$$(n\ f_2(b)) \cdot (f_1(b)) = f_1((n\ f_2(b)) \cdot b) = f_1((n\ f_2(b')) \cdot b') = (n\ f_2(b')) \cdot (f_1(b')). \tag{3.10}$$

Since the codomain of f is $\Gamma * \mathbf{N}$, we have $f_1(b)\ \#\ f_2(b)$. Since f_1 is equivariant, it follows from $n\ \#\ b$ that $n\ \#\ f_1(b)$ holds. Therefore, we have $(n\ f_2(b)) \cdot (f_1(b)) = f_1(b)$. By similar reasoning we obtain $(n\ f_2(b')) \cdot (f_1(b')) = f_1(b')$. With the above equation (3.10), this implies the required equation $f_1(b) = f_1(b')$. Well-definedness of the morphism part follows similarly.

We now show that $\Sigma^*_{\mathbf{N}}$ is a fibred right adjoint to W_A. First we have to show that $\Sigma^*_{\mathbf{N}}$ is a fibred functor.

Lemma 3.3.22. *The functor $\Sigma^*_{\mathbf{N}}$ as defined above is a fibred functor from $Gl(-*A)$ to cod.*

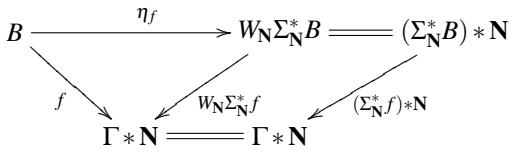
Proof. It is easy to see from the definition that $\Sigma^*_{\mathbf{N}}$ makes diagram (3.4) commute. It remains to check that $\Sigma^*_{\mathbf{N}}$ preserves cartesian maps, which in this case requires to show that if (3.8) is a pullback then so is (3.9). We describe the main proof idea informally; full details can be found in the proof of Lemma 3.3.31. Assuming (3.8) to be a pullback means that, for all $\langle \gamma, n\rangle \in \Gamma * \mathbf{N}$ and all $c \in C$ with $\langle v(\gamma), n\rangle = g(c)$, there exists a unique $b \in B$ satisfying $u(b) = c$ and $f(b) = \langle \gamma, n\rangle$. To show that (3.9) is a pullback, we have to show that, for all $\gamma \in \Gamma$ and all $[c]_g \in \Sigma^*_{\mathbf{N}}C$ satisfying $v(\gamma) = (\Sigma^*_{\mathbf{N}}g)([c]_g)$, there exists a unique $[b]_f \in \Sigma^*_{\mathbf{N}}B$ satisfying $(\Sigma^*_{\mathbf{N}}u)([b]_f) = [c]_g$ and $(\Sigma^*_{\mathbf{N}}f)([b]_f) = \gamma$. Given $\gamma \in \Gamma$ and $[c]_g \in \Sigma^*_{\mathbf{N}}C$, we would like to use the pullback (3.8) for $\langle \gamma, g_2(c)\rangle$ and c. However, this does not quite work since $\langle \gamma, g_2(c)\rangle$ is an element of $\Gamma \times \mathbf{N}$ but not necessarily of $\Gamma * \mathbf{N}$. To use the pullback (3.8) we must find a representative c' of the class $[c]_g$ such that $\langle \gamma, g_2(c')\rangle \in \Gamma * \mathbf{N}$ holds. Such an element c' can be found by freshening c, that is by letting $c' \stackrel{\text{def}}{=} (n\ g_2(c)) \cdot g(c)$ for some fresh name n. With this definition, we can now use the pullback (3.8) to obtain the required element.

In essence, the construction of c' by freshening c amounts to the usual practice that the bound name in an α-equivalence is always assumed to be fresh. □

Proposition 3.3.23. *The functor $\Sigma^*_{\mathbf{N}}$ is a fibred left adjoint to $W_{\mathbf{N}}$.*

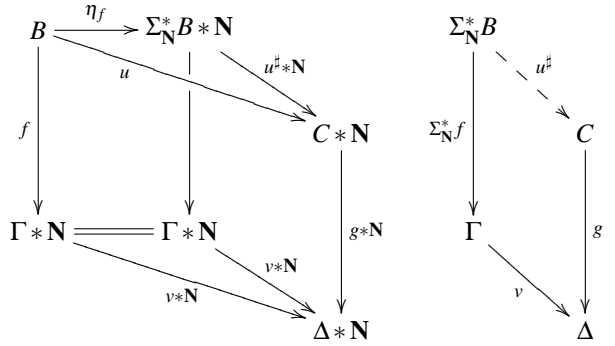
Proof. We have already checked that both functors are fibred functors. Therefore, it suffices to show an adjunction $\Sigma^*_{\mathbf{N}} \dashv W_{\mathbf{N}}$ with a vertical unit. Let $f\colon B \to \Gamma * \mathbf{N}$ be an object of $\mathbb{S}/(-*\mathbf{N})$. We define the

vertical unit η_f of the adjunction



by $\eta_f(b) = \langle [b]_f, f_2(b)\rangle$. This pair is an element of $(\Sigma^*_{\mathbf{N}}B)*\mathbf{N}$ since we have $[b]_f \# f_2(b)$, which follows just as in Corollary 5.2 of [38]. It is straightforward to check that this defines a natural transformation $\eta : Id \to W_{\mathbf{N}}\Sigma^*_{\mathbf{N}}$.

For universality, we have to show that for any morphism $(u,v)\colon f \to W_{\mathbf{N}}g$ in $\mathbb{S}/(-*\mathbf{N})$ there exists a unique morphism $(u^\sharp, v)\colon \Sigma^*_{\mathbf{N}}f \to g$ in $\mathbb{S}^\to$ with $(u^\sharp * \mathbf{N}, v)\circ(\eta_f, id) = (u,v)$, as in the following diagram.



If it exists, the map $u^\sharp$ necessarily satisfies $u^\sharp([b]_f) = \dot\pi_1(u(b))$ and is therefore unique. To show existence, it suffices to verify that $u^\sharp([b]_f) = \dot\pi_1(u(b))$ defines an equivariant function, i.e. that $b \sim_f b'$ implies $\dot\pi_1(u(b)) = \dot\pi_2(u(b))$. Suppose $b \sim_f b'$, which means that there exists a fresh name n such that $(n\ f_2(b))\cdot b = (n\ f_2(b'))\cdot b'$. By equivariance of $\dot\pi_1 \circ u$, this implies $(n\ f_2(b))\cdot(\dot\pi_1(u(b))) = \dot\pi_1(u((n\ f_2(b))\cdot b)) = \dot\pi_1(u((n\ f_2(b'))\cdot b')) = (n\ f_2(b'))\cdot(\dot\pi_1(u(b')))$. Since the left square of the above diagram commutes, we have $\dot\pi_2(u(b)) = f_2(b)$. By the codomain of u this implies $\dot\pi_1(u(b)) \# f_2(b)$, and similarly $\dot\pi_1(u(b')) \# f_2(b')$. Since n was chosen freshly, we therefore have $\dot\pi_1(u(b)) = (n\ f_2(b))\cdot(\dot\pi_1(u(b))) = (n\ f_2(b'))\cdot(\dot\pi_1(u(b'))) = \dot\pi_1(u(b'))$. Showing equivariance of $u^\sharp$ is routine. □

We spell out the construction of $\Sigma^*_{\mathbf{N}}A$ for a non-dependent type A, in order to show that it specialises to the abstraction set of Gabbay and Pitts [38]. Let A be an object of $\mathbb{S}$. This object amounts to a closed type $\vdash A$ Type. By weakening, we obtain a type $\diamond * (n\colon \mathbf{N}) \vdash A$ Type. It corresponds to the object $f\colon (1*\mathbf{N})\times A \to (1*\mathbf{N})$ of $\mathbb{S}/(1*\mathbf{N})$ given by the first projection. Applying $\Sigma^*_{\mathbf{N}}$ to this object gives us an object $\Sigma^*_{\mathbf{N}}f$ in $\mathbb{S}/1$. We spell out $\Sigma^*_{\mathbf{N}}f$ to show that it is essentially the same as the abstraction set $[\mathbf{N}]A$ of [38]. First we consider the equivalence classes of $\sim_f$. The relation $\sim_f$ is an equivalence relation on $(1*\mathbf{N})\times A$, so that all its equivalence classes have the form $[\langle\langle\diamond, n\rangle, a\rangle]_f$ for some $n \in \mathbf{N}$ and $a \in A$.

Unfolding the definitions, and using (34) of [38] in the last step, gives:

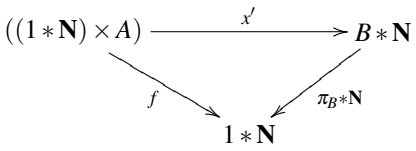
$$\begin{aligned}[\langle\langle\diamond,n\rangle,a\rangle]_f &= \{\langle\langle\diamond,n'\rangle,a'\rangle \mid \langle\langle\diamond,n\rangle,a\rangle \sim_f \langle\langle\diamond,n'\rangle,a'\rangle\}\\ &= \{\langle\langle\diamond,n'\rangle,a'\rangle \mid \exists m.\,(m \# \langle n,a,n',a'\rangle) \wedge (m\ n)\cdot\langle\langle\diamond,n\rangle,a\rangle = (m\ n')\cdot\langle\langle\diamond,n'\rangle,a'\rangle\}\\ &= \{\langle\langle\diamond,n'\rangle,a'\rangle \mid \exists m.\,(m \# \langle n,a,n',a'\rangle) \wedge (m\ n)\cdot a = (m\ n')\cdot a'\}\\ &= \{\langle\langle\diamond,n'\rangle,a'\rangle \mid a' = (n\ n')\cdot a \wedge (n = n' \vee n' \# a)\}\end{aligned}$$

This means that $[\langle\langle\diamond,n\rangle,a\rangle]_f$ is, up to the superfluous component $\diamond$, the same as the abstraction $n.a$ defined in (35) of [38]. Next we spell out the domain of $\Sigma^*_{\mathbf{N}} f\colon \Sigma^*_{\mathbf{N}}((1*\mathbf{N})\times A) \to 1$. It is defined to be

$$\Sigma^*_{\mathbf{N}}((1*\mathbf{N})\times A) = \{[\langle\langle\diamond,n\rangle,a\rangle]_f \mid n \in \mathbf{N} \wedge a \in A\}.$$

This amounts to the definition of the abstraction set $[\mathbf{N}]A$ in [38], again up to an inessential component of unit type.

The adjunction $\Sigma^*_{\mathbf{N}} \dashv W_{\mathbf{N}}$ captures familiar constructions from [38]. The unit $\eta_f\colon f \to W_{\mathbf{N}}\Sigma^*_{\mathbf{N}} f$ in $\mathbb{S}/(1*\mathbf{N})$ is defined by $\eta_f(\langle\langle\diamond,n\rangle,a\rangle) = \langle[\langle\langle\diamond,n\rangle,a\rangle]_f, n\rangle \in (\Sigma^*_{\mathbf{N}}((1*\mathbf{N})\times A))*\mathbf{N}$. In this way, the unit η contains the abstraction operation of [38], which maps n and a to $n.a$. The unit also contains the freshness information $n \# n.a$. In Lemma 6.3 of [38], a universal property of the abstraction set is shown. This property corresponds to the universality of the unit of the adjunction $\Sigma^*_{\mathbf{N}} \dashv W_{\mathbf{N}}$. For simplicity, we consider just the special case without a parameter context. In this case, Lemma 6.3 of [38] states that for any map $x\colon \mathbf{N}\times A \to B$ in $\mathbb{S}$ satisfying $x(n,a) \# n$ for all $n \in \mathbf{N}$ and all $a \in A$, there exists a unique map $x^\sharp\colon [\mathbf{N}]A \to B$ in $\mathbb{S}$ such that $x(n,a) = x^\sharp(n.a)$ holds for all $n \in \mathbf{N}$ and all $a \in A$. A map $x\colon \mathbf{N}\times A \to B$ in $\mathbb{S}$ for which $x(n,a) \# n$ holds for all $n \in \mathbf{N}$ and $a \in A$ corresponds uniquely to a map x' in $\mathbb{S}/(1*\mathbf{N})$ as in the diagram below. This can be seen by letting $x'(\langle\langle\diamond,n\rangle,a\rangle) = \langle x(n,a), n\rangle$.



The codomain of x' is $\pi_B * \mathbf{N} = W_{\mathbf{N}}\pi_B$. By the adjunction $\Sigma^*_{\mathbf{N}} \dashv W_{\mathbf{N}}$, the map x' corresponds uniquely to a map $x'^\sharp\colon \Sigma^*_{\mathbf{N}} f \to \pi_B$ in $\mathbb{S}/1$ satisfying $x'^\sharp([\langle\langle\diamond,n\rangle,a\rangle]_f) = \pi_1(x'(\langle\langle\diamond,n\rangle,a\rangle))$. In terms of Lemma 6.3 of [38], this corresponds to the existence of a function $x^\sharp$ that satisfies $x^\sharp(n.a) = x(n,a)$. The more general case with a parameter context follows similarly by using $\Gamma * \mathbf{N}$ instead of $1*\mathbf{N}$.

We remark that not all the constructions that are used for abstraction sets in [38] are captured by the adjunction $\Sigma^*_{\mathbf{N}} \dashv W_{\mathbf{N}}$. For example, the concretion operation $(n.a)@m$ of [38] is not given by this adjunction. In Chapter 10, we show how concretion can nevertheless be captured in terms of Σ^* and Π^*.

In the rest of this section we define Σ^*_A for an arbitrary object A. This definition directly generalises the definition of $\Sigma^*_{\mathbf{N}}$.

Definition 3.3.24. Let X and Y be finite sets of names. Define the set $(X \leftrightarrow Y) \subseteq Aut(\mathbf{N})$ of *swappings of X and Y* to consist of all permutations $\pi \in Aut(\mathbf{N})$ that satisfy $\pi(X) = Y$, $\pi(Y) = X$, and, for all $z \in \mathbf{N} \setminus (X \cup Y)$, $\pi(z) = z$.

Lemma 3.3.25. *For any two finite sets of names X and Y, the set $(X{\leftrightarrow}Y)$ is not empty if and only if $|X| = |Y|$ holds.*

We omit the straightforward proof.

For $f\colon B \to \Gamma * A$ we define the relation $\sim_f \subseteq B \times B$ by

$$b \sim_f b' \iff \begin{array}{l} \exists X \in \mathscr{P}_{\mathrm{fin}}(\mathbf{N}).\ X \# \langle b, b' \rangle \wedge \\ \qquad \exists \pi \in (\mathrm{supp}(f_2(b)){\leftrightarrow}X).\ \exists \pi' \in (\mathrm{supp}(f_2(b')){\leftrightarrow}X).\ \pi \cdot b = \pi' \cdot b'. \end{array} \tag{3.11}$$

In the next four lemmas, we generalise the results of [38, §5] on name abstraction.

Lemma 3.3.26. *For all $b, b' \in B$ we have*

$$b \sim_f b' \iff \begin{array}{l} \forall X \in \mathscr{P}_{|\mathrm{supp}(f_2(b))|}(\mathbf{N}).\ X \# \langle b, b' \rangle \implies \\ \qquad \exists \pi \in (\mathrm{supp}(f_2(b)){\leftrightarrow}X).\ \exists \pi' \in (\mathrm{supp}(f_2(b')){\leftrightarrow}X).\ \pi \cdot b = \pi' \cdot b', \end{array} \tag{3.12}$$

where $\mathscr{P}_{|\mathrm{supp}(f_2(b))|}(\mathbf{N})$ is the set of all subsets of $\mathbf{N}$ having the same number of elements as $\mathrm{supp}(f_2(b))$.

Proof. The implication from right to left holds because by the finite support property we can always find a set $X \in \mathscr{P}_{|\mathrm{supp}(f_2(b))|}(\mathbf{N})$ of names that is fresh for both b and b'.

From left to right, suppose we have $X \in \mathscr{P}_{\mathrm{fin}}(\mathbf{N})$, $\pi \in (\mathrm{supp}(f_2(b)){\leftrightarrow}X)$ and $\pi' \in (\mathrm{supp}(f_2(b')){\leftrightarrow}X)$ with $X \# \langle b, b' \rangle$ and $\pi \cdot b = \pi' \cdot b'$. Lemma 3.3.25 implies $|X| = |\mathrm{supp}(f_2(b))| = |\mathrm{supp}(f_2(b'))|$. Let $Y \in \mathscr{P}_{|\mathrm{supp}(f_2(b))|}(\mathbf{N})$ with $Y \# \langle b, b' \rangle$. By Lemma 3.3.25, there exists $\tau \in (X{\leftrightarrow}Y)$. By definition of the set $(X{\leftrightarrow}Y)$, we have $(\tau \circ \pi)(\mathrm{supp}(f_2(b))) = Y$. Since Y consists of names that are fresh for b and b', the definition

$$\kappa(n) = \begin{cases} (\tau \circ \pi)(n) & \text{if } n \in \mathrm{supp}(f_2(b)) \\ (\tau \circ \pi)^{-1}(n) & \text{if } n \in Y \\ n & \text{otherwise} \end{cases}$$

therefore defines an element of $(\mathrm{supp}(f_2(b)){\leftrightarrow}Y)$. Notice that $\kappa(n) = (\tau \circ \pi)(n)$ holds for all $n \in \mathrm{supp}(b)$. This implies $\kappa \cdot b = (\tau \circ \pi) \cdot b$. Similarly, we define $\kappa' \in (\mathrm{supp}(f_2(b')){\leftrightarrow}Y)$ such that $\kappa' \cdot b' = (\tau \circ \pi') \cdot b'$ holds. From $\pi \cdot b = \pi' \cdot b'$ we therefore get $\kappa \cdot b = \tau \cdot \pi \cdot b = \tau \cdot \pi' \cdot b' = \kappa' \cdot b'$, as required. □

Lemma 3.3.27. *For any $f\colon B \to \Gamma * A$, the relation $\sim_f$ is an equivariant equivalence relation.*

Proof. Reflexivity and symmetry are immediate. Transitivity follows using Lemma 3.3.26. Verifying equivariance is straightforward. For example, it can be seen because $\sim_f$ can be defined in the internal logic of $\mathbb{S}$ as a subobject of $B \times B$, and so equivariance holds automatically. □

As before, we write $[b]_f$ for the equivalence class of $b \in B$ under $\sim_f$.

Lemma 3.3.28. *For all $f\colon B \to \Gamma * A$ and $b \in B$, the following equivalence holds.*

$$b \sim_f b' \iff \exists \pi.\ (\forall n \in \mathrm{supp}(b) \setminus \mathrm{supp}(f_2(b)).\ \pi(n) = n) \wedge \pi \cdot b = b'$$

Proof. The proof is a direct generalisation of the proof of Lemma 5.1 in [38].

From left to right, assume $b \sim_f b'$. Write X for $\mathrm{supp}(f_2(b))$ and X' for $\mathrm{supp}(f_2(b'))$. By definition of $\sim_f$, there exist a finite set of fresh names Y and permutations $\tau \in X \leftrightarrow Y$ and $\tau' \in X' \leftrightarrow Y$ such that $\tau \cdot b = \tau' \cdot b'$ holds.

First we show $X' \cap (\mathrm{supp}(b) \setminus X) = \emptyset$. Suppose $n \in X'$. We have

$$\begin{aligned}
&\tau'^{-1}(n) \# b' && \text{since } \tau'(Y) = X' \text{ and all names in } Y \text{ are fresh for } b' \\
&\implies \tau' \cdot \tau'^{-1} \cdot n \# \tau' \cdot b' && \text{by Proposition 3.3.11} \\
&\implies n \# \tau \cdot b && \text{since } \tau \cdot b = \tau' \cdot b' \\
&\implies \tau^{-1} \cdot n \# \tau^{-1} \cdot \tau \cdot b && \text{by Proposition 3.3.11} \\
&\implies \tau^{-1}(n) \# b
\end{aligned}$$

Since $\tau \in X \leftrightarrow Y$, it follows that $\tau^{-1}(n) \# b$ implies $(n \in X) \vee (n \# b)$. Thus, we have shown that all $n \in X'$ satisfy $(n \in X) \vee (n \# b)$. In other words, $X' \cap (\mathrm{supp}(b) \setminus X) = \emptyset$.

Now consider the permutation $\tau'^{-1} \circ \tau$. It maps X to X'. Hence, there exists a permutation $\pi \in X \leftrightarrow X'$ such that $\pi(n) = (\tau'^{-1} \circ \tau)(n)$ holds for all $n \in X$. By definition, π is the identity on at least $\mathbf{N} \setminus (X \cup X')$ and $\tau'^{-1} \circ \tau$ is the identity on at least $\mathbf{N} \setminus (Y \cup X \cup X')$. Since $X' \cap (\mathrm{supp}(b) \setminus X) = \emptyset$ and Y is fresh for b, both permutations are the identity on $\mathrm{supp}(b) \setminus X$. Moreover, since by definition both permutations agree on X, they therefore agree on the whole of $\mathrm{supp}(b)$. Hence, $\pi \cdot b = (\tau'^{-1} \circ \tau) \cdot b$ holds. Since we have $(\tau'^{-1} \circ \tau) \cdot b = \tau'^{-1} \cdot \tau \cdot b = \tau'^{-1} \cdot \tau' \cdot b' = b'$, this implies $\pi \cdot b = b'$. Therefore, π is a permutation as required to instantiate the existential quantifier in the left-to-right direction.

It remains to show the right-to-left implication. Let π be a permutation satisfying both $\pi \cdot b = b'$ and $(\forall n \in \mathrm{supp}(b) \setminus \mathrm{supp}(f_2(b)).\ \pi(n) = n)$. We have to show $b \sim_f b'$. Write X for $\mathrm{supp}(f_2(b))$ and X' for $\mathrm{supp}(f_2(b'))$. Observe that $\pi \cdot b = b'$ implies $|X| = |X'|$. Let Y be a set of $|X|$ fresh names. There exist permutations $\tau \in X \leftrightarrow Y$ and $\tau' \in X' \leftrightarrow Y$ such that $\pi(n) = (\tau'^{-1} \circ \tau)(n)$ holds for all $n \in X$. For $b \sim_f b'$ it suffices to show $\tau \cdot b = \tau' \cdot b'$. We now show that π and $(\tau'^{-1} \circ \tau)$ agree on $\mathrm{supp}(b)$. Since by definition they agree on X, it just remains to show that they agree on $\mathrm{supp}(b) \setminus X$. By assumption, we have $\forall n \in \mathrm{supp}(b) \setminus X.\ \pi(n) = n$. Since $X' = \pi(X)$, this implies $(\mathrm{supp}(b) \setminus X) \cap X' = \emptyset$. Since Y consists of fresh names, $(\tau'^{-1} \circ \tau)$ is also the identity on $\mathrm{supp}(b) \setminus X$. Hence π and $(\tau'^{-1} \circ \tau)$ agree on $\mathrm{supp}(b)$. From this we get $\pi \cdot b = (\tau'^{-1} \circ \tau) \cdot b$. But with $\pi \cdot b = b'$, this implies $b' = (\tau'^{-1} \circ \tau) \cdot b$, and thus the required $\tau' \cdot b' = \tau \cdot b$. □

Corollary 3.3.29. *For all $f\colon B \to \Gamma * A$ and $b \in B$, the equivalence classes of $\sim_f$ have the following characterisation.*

$$[b]_f = \{\pi \cdot b \mid \forall n \in \mathrm{supp}(b) \setminus \mathrm{supp}(f_2(b)).\ \pi(n) = n\}$$

Corollary 3.3.30. *For all $f\colon B \to \Gamma * A$ and $b \in B$, we have* $\mathrm{supp}([b]_f) = \mathrm{supp}(b) \setminus \mathrm{supp}(f_2(b))$.

Proof. By Corollary 3.3.29, any permutation that fixes all names in $\mathrm{supp}(b) \setminus \mathrm{supp}(f_2(b))$ also fixes $[b]_f$. Hence, $\mathrm{supp}([b]_f) \subseteq \mathrm{supp}(b) \setminus \mathrm{supp}(f_2(b))$. On the other hand, $\mathrm{supp}(\pi \cdot b) = \pi(\mathrm{supp}(b))$ holds for

all π and b. Therefore, Corollary 3.3.29 implies $\mathrm{supp}(b) \setminus \mathrm{supp}(f_2(b)) \subseteq \mathrm{supp}(b')$ for all $b' \in [b]_f$. Lemma 3.3.16 gives the required $\mathrm{supp}(b) \setminus \mathrm{supp}(f_2(b)) \subseteq \mathrm{supp}([b]_f)$. □

For any object A, we now define the fibred functor Σ_A^*.

$$\begin{array}{ccc} \mathbb{S}/(-*A) & \xrightarrow{\Sigma_A^*} & \mathbb{S}^{\to} \\ & {\scriptstyle Gl(-*A)} \searrow \quad \swarrow {\scriptstyle cod} & \\ & \mathbb{S} & \end{array} \tag{3.13}$$

It maps an object $f\colon B \to \Gamma * A$ to the object $\Sigma_A^* f\colon \Sigma_A^* B \to \Gamma$ that is defined by $\Sigma_A^* B \stackrel{\mathrm{def}}{=} \{[b]_f \mid b \in B\}$, $\pi \cdot_{\Sigma_A^* B} [b]_f \stackrel{\mathrm{def}}{=} [\pi \cdot_B b]_f$ and $(\Sigma_A^* f)([b]_f) \stackrel{\mathrm{def}}{=} f_1(b)$. A morphism $(u,v)\colon f \to g$ in $\mathbb{S}/(-*A)$ as given by

$$\begin{array}{ccc} B & \xrightarrow{u} & C \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ \Gamma * A & \xrightarrow[v*A]{} & \Delta * A \end{array} \tag{3.14}$$

is mapped by Σ_A^* to the morphism

$$\begin{array}{ccc} \Sigma_A^* B & \xrightarrow{\Sigma_A^* u} & \Sigma_A^* C \\ {\scriptstyle \Sigma_A^* f}\downarrow & & \downarrow{\scriptstyle \Sigma_A^* g} \\ \Gamma & \xrightarrow[v]{} & \Delta \end{array} \tag{3.15}$$

defined by $(\Sigma_A^* u)([b]_f) = [u(b)]_g$. Well-definedness follows as for $\Sigma_{\mathbf{N}}^*$.

Lemma 3.3.31. *The functor Σ_A^* is a fibred functor from $Gl(-*A)$ to cod.*

Proof. That Σ_A^* makes diagram (3.13) commute follows directly from the definition. It remains to check that Σ_A^* preserves cartesian maps. We have to show that if (3.14) is a pullback then so is (3.15). To this end, we can assume that, for all $\langle \gamma, a\rangle \in \Gamma * A$ and all $c \in C$ with $\langle v(\gamma), a\rangle = g(c)$, there exists a unique $b \in B$ with $u(b) = c$ and $f(b) = \langle \gamma, a\rangle$. We have to show that, for all $\gamma \in \Gamma$ and all $[c]_g \in \Sigma_A^* C$ with $v(\gamma) = (\Sigma_A^* g)([c]_g)$, there exists a unique $[b]_f \in \Sigma_A^* B$ with $(\Sigma_A^* u)([b]_f) = [c]_g$ and $(\Sigma_A^* f)([b]_f) = \gamma$.

Let $\gamma \in \Gamma$ and $[c]_g \in \Sigma_A^* C$ with $v(\gamma) = (\Sigma_A^* g)([c]_g)$. We first show the existence of $[b]_f$ with the required properties. Consider $g_2(c)$ and notice that $g_2(c)$ need not be fresh for γ. However, we can always find a set $X \in \mathscr{P}_{|\mathrm{supp}(g_2(c))|}(\mathbf{N})$ of fresh names and a permutation $\pi \in X \leftrightarrow \mathrm{supp}(g_2(c))$. By definition of $\sim_g$, we have $[\pi \cdot c]_g = [c]_g$. Since g has codomain $\Delta * A$, we have $g_1(c) \# g_2(c)$. Since π fixes all names not in $X \cup \mathrm{supp}(g_2(c))$, and X consists of fresh names only, this implies $\pi \cdot g_1(c) = g_1(c)$. Hence, we have $g(\pi \cdot c) = \pi \cdot g(c) = \pi \cdot \langle g_1(c), g_2(c)\rangle = \langle g_1(c), \pi \cdot g_2(c)\rangle = \langle (\Sigma_A^* g)([c]_g), \pi \cdot g_2(c)\rangle = \langle v(\gamma), g_2(\pi \cdot c)\rangle$. Since $\langle \gamma, g_2(\pi \cdot c)\rangle \in \Gamma * A$, we can apply the assumption to get an element $b \in B$ that satisfies $f(b) = \langle \gamma, g_2(\pi \cdot c)\rangle$ and $u(b) = \pi \cdot c$. This gives $(\Sigma_A^* u)([b]_f) = [u(b)]_g = [\pi \cdot c]_g = [c]_g$ and $(\Sigma_A^* f) = f_1(b) = \gamma$, as required.

For uniqueness, let $\gamma \in \Gamma$ and $[c]_g \in \Sigma^*_{\mathbf{N}} C$ with $v(\gamma) = (\Sigma^*_{\mathbf{N}} g)([c]_g)$, and suppose that there are $b, b' \in B$ such that both $(\Sigma^*_{\mathbf{N}} u)([b]_f) = [c]_g = (\Sigma^*_{\mathbf{N}} u)([b']_f)$ and $(\Sigma^*_{\mathbf{N}} f)([b]_f) = \gamma = (\Sigma^*_{\mathbf{N}} f)([b']_f)$ hold. We have to show $[b]_f = [b']_f$. Since, by definition, $(\Sigma^*_{\mathbf{N}} u)([b]_f) = [u(b)]_g$ and $(\Sigma^*_{\mathbf{N}} u)([b']_f) = [u(b')]_g$ hold, we have $[u(b)]_g = [c]_g = [u(b')]_g$, and thus $u(b) \sim_g u(b')$. By definition of $\sim_g$ this implies $|\mathrm{supp}(u(b))| = |\mathrm{supp}(u(b'))|$. Let X be a set of fresh names of size $|\mathrm{supp}(g_2(u(b)))| = |\mathrm{supp}(g_2(u(b')))|$. By (3.12) we have permutations $\pi \in \mathrm{supp}(g_2(u(b))) \leftrightarrow X$ and $\pi' \in \mathrm{supp}(g_2(u(b'))) \leftrightarrow X$ for which $\pi \cdot u(b) = \pi' \cdot u(b')$ holds. Define $d \stackrel{\text{def}}{=} \pi \cdot u(b)$ and $a \stackrel{\text{def}}{=} g_2(\pi \cdot u(b)) = g_2(\pi \cdot u(b'))$. By definition of π and equivariance, we obtain $\mathrm{supp}(a) = X$, which implies $\langle \gamma, a \rangle \in \Gamma * A$ since all names in X are fresh. We apply the pullback (3.14) to $d \in C$ and $\langle \gamma, a \rangle \in \Gamma * A$. The following reasoning shows that it is applicable.

$$
\begin{aligned}
g(d) &= g(\pi \cdot u(b)) \\
&= \langle g_1(\pi \cdot u(b)), g_2(\pi \cdot u(b)) \rangle \\
&= \langle \pi \cdot g_1(u(b)), a \rangle \\
&= \langle g_1(u(b)), a \rangle && \text{since } g_1(u(b)) \,\#\, g_2(u(b)) = a \text{ and } \pi \in \mathrm{supp}(g_2(u(b))) \leftrightarrow X \\
&= \langle (\Sigma^*_{\mathbf{N}} g)([u(b)]_g), a \rangle && \text{by definition} \\
&= \langle (\Sigma^*_{\mathbf{N}} g)([c]_g), a \rangle && \text{by } [u(b)]_g = [c]_g \\
&= \langle g_1(c), a \rangle && \text{by definition} \\
&= \langle v(\gamma), a \rangle
\end{aligned}
$$

Using pullback (3.14), there exists a unique $b_0 \in B$ satisfying $u(b_0) = d$ and $f(b_0) = \langle \gamma, a \rangle$. We show that both $\pi \cdot b$ and $\pi' \cdot b'$ have this property. First, we have $u(\pi \cdot b) = \pi \cdot u(b) = d$ by equivariance and definition. To show $f(\pi \cdot b) = \langle \gamma, a \rangle$ we first show $\pi \cdot f_1(b) = f_1(b)$. To this end, we have $f_1(b) \,\#\, f_2(b)$ by the form of the codomain of f. Since (3.14) commutes, we have $f_2(b) = g_2(u(b))$, and this implies $f_1(b) \,\#\, g_2(u(b))$. By definition, π fixes all names not in $X \cup \mathrm{supp}(g_2(u(b)))$. Since X contains only fresh names, this implies that π fixes all names in $\mathrm{supp}(f_1(b))$. Hence, we have $\pi \cdot f_1(b) = f_1(b)$. The equation $f(b_1) = \langle \gamma, a \rangle$ can now be shown as follows.

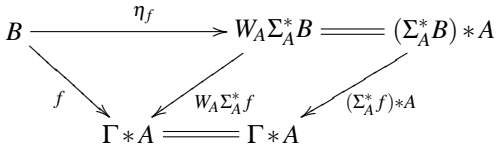
$$
\begin{aligned}
f(\pi \cdot b) = \pi \cdot f(b) &= \pi \cdot \langle f_1(b), f_2(b) \rangle \\
&= \pi \cdot \langle f_1(b), g_2(u(b)) \rangle && \text{since (3.14) commutes} \\
&= \langle f_1(b), a \rangle && \text{using } \pi \cdot f_1(b) = f_1(b) \\
&= \langle (\Sigma^*_{\mathbf{N}} f)([b]_f), a \rangle && \text{by definition} \\
&= \langle \gamma, a \rangle && \text{by assumption } \gamma = (\Sigma^*_{\mathbf{N}} f)([b]_f)
\end{aligned}
$$

The equations $u(\pi' \cdot b') = d$ and $f(\pi' \cdot b') = \langle \gamma, a \rangle$ follow analogously. By uniqueness of b_0, this implies $\pi \cdot b = b_0 = \pi' \cdot b'$. Using (3.11) we obtain $b \sim_f b'$, and thus the required $[b]_f = [b']_f$. □

Proposition 3.3.32. *The functor Σ^*_A is a fibred left adjoint to W_A.*

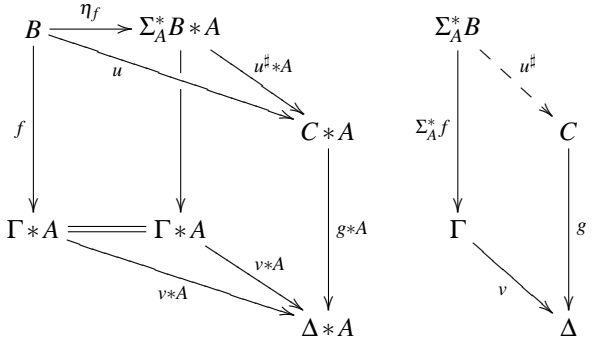
Proof. We have already checked that both functors are fibred functors. Therefore, it suffices to show an adjunction $\Sigma^*_A \dashv W_A$ with a vertical unit. Let $f \colon B \to \Gamma * A$ be an object of $\mathbb{S}/(- * A)$. We define the

vertical unit η_f of the adjunction



by $\eta_f(b) = \langle [b]_f, f_2(b)\rangle$. This pair is an element of $(\Sigma_A^*B)*A$ since $[b]_f \# f_2(b)$ holds by Corollary 3.3.30. It is straightforward to see that this defines a natural transformation $\eta : Id \to W_A\Sigma_A^*$.

For universality, we have to show that for any morphism $(u,v) : f \to W_A g$ in $\mathbb{S}/(-*A)$ there exists a unique morphism $(u^\sharp, v) : \Sigma_A^* f \to g$ in $\mathbb{S}^\to$ satisfying $(u^\sharp * A, v) \circ (\eta_f, id) = (u,v)$, as shown in the following diagram.



If it exists, the map $u^\sharp$ necessarily satisfies $u^\sharp([b]_f) = \dot{\pi}_1(u(b))$ and is therefore unique. To show existence, it suffices to verify that $u^\sharp([b]_f) = \dot{\pi}_1(u(b))$ defines an equivariant function, i.e. that $b \sim_f b'$ implies $\dot{\pi}_1(u(b)) = \dot{\pi}_1(u(b'))$. Suppose $b \sim_f b'$. Then there exists a set X of fresh names and permutations $\pi : X \leftrightarrow \mathrm{supp}(f_2(b))$ and $\pi' : X \leftrightarrow \mathrm{supp}(f_2(b'))$ such that $\pi \cdot b = \pi' \cdot b'$ holds. By equivariance of $\dot{\pi}_1 \circ u$, this implies $\pi \cdot (\dot{\pi}_1(u(b))) = \dot{\pi}_1(u(\pi \cdot b)) = \dot{\pi}_1(u(\pi' \cdot b')) = \pi' \cdot (\dot{\pi}_1(u(b')))$. Since the left square in the above diagram commutes, we have $\dot{\pi}_2(u(b)) = f_2(b)$. By the form of the codomain of u this implies $\dot{\pi}_1(u(b)) \# f_2(b)$, and similarly $\dot{\pi}_1(u(b')) \# f_2(b')$. Since all the names in X are fresh, we therefore have $\dot{\pi}_1(u(b)) = \pi \cdot (\dot{\pi}_1(u(b))) = \pi' \cdot (\dot{\pi}_1(u(b'))) = \dot{\pi}_1(u(b'))$. Equivariance of $u^\sharp$ follows straightforwardly. □

3.4 The Realizability Category $\mathbf{Ass}_S(P)$

In the previous section we have introduced the Schanuel topos and constructed Π^* and Σ^* for it. If we want to use these constructs for programming with names and binding, we must give an effectively computable version of them. In this section we define the realizability category $\mathbf{Ass}_S(P)$, an effective version of the Schanuel topos.

To endow a Schanuel sets with computability information, we proceed in a similar way to the construction of assemblies over sets, see e.g. [61]. Assume for the moment that our effective programming

language has natural numbers as data and that programs are partial computable functions on natural numbers. To make a Schanuel set into a data type in this programming language, we equip each of its elements with a non-empty set of natural numbers consisting of the codes representing that element. We are then interested in computable morphisms between such sets, by which we mean morphisms $f\colon A \to B$ for which there is a program that maps each code for $a \in A$ to a code for $f(a) \in B$. So far, this is just the well-known description of assemblies on sets. For example, the cartesian closed structure of assemblies is such that $A \Rightarrow B$ can be described as the set of all functions from $A \Rightarrow B$ that are computed by some program. In an effective version of the Schanuel topos, we should expect $A \Rightarrow B$ to consist of all computable functions with finite support, equipped with the action $\pi \cdot f = \lambda a.\, \pi \cdot f(\pi^{-1} \cdot a)$. Notice that the so defined function $\pi \cdot f$ has to be computable as well. Therefore, it is natural not only to equip the underlying set of a Schanuel set with codes, but also to require the permutation action on the Schanuel set to be computable. For this to make sense, we must also equip the group of permutations with computability information.

This motivates the construction of $\mathbf{Ass}_S(P)$ as a category of internal group actions in the category **Ass** of assemblies. An internal group object P in **Ass** is a group whose underlying set is endowed with codes and whose group operations are computable. A left P-action in **Ass** is a pair $(A, \mu_A\colon P \times A \to A)$ satisfying appropriate conditions. Such a pair amounts to a set A equipped with codes and a computable action μ_A on A. Since the Schanuel topos is defined similarly using actions in **Sets**, we may expect this definition to give a computable version of the Schanuel topos.

Our presentation of $\mathbf{Ass}_S(P)$ is external in the sense that we work directly with assemblies and their codes rather than using an internal language. This has the advantage that it is immediately clear what the various constructions amount to in terms of the realizers, thus showing that the definitions correspond to computational intuitions. One disadvantage is that it only gives us a quasi-topos, not a topos. A potential route to obtaining a topos is to carry out the construction in the internal language of a realizability topos, such as the effective topos **Eff** [52]. A stumbling block on this route is the fact that the internal logic of the category of P-actions inside **Eff** is not classical; and the definition of support in the Schanuel topos makes use of classical logic. It would nevertheless be interesting to see if the construction of an effective version of the Schanuel topos can be carried out in the internal language of a realizability topos and how such a construction relates to the category $\mathbf{Ass}_S(P)$ that we are about to define. Since the structure of a quasi-topos is completely sufficient for our purposes, we settle with the external development, leaving the internal development for future work.

3.4.1 Assemblies

We introduce the category of assemblies on sets following [61, 62], where a more in-depth study of this category can be found.

Definition 3.4.1. A *partial applicative structure* $(\mathscr{X}, \cdot)$ is a set $\mathscr{X}$ with a partial operation $\cdot$ from $\mathscr{X} \times \mathscr{X}$ to $\mathscr{X}$.

For working with a partial applicative structure, we use *formal expressions*. A formal expression on $(\mathscr{X},\cdot)$, is either a variable x, an element $r \in \mathscr{X}$ or a formal application $e\ e'$ where both e and e' are formal expressions on $(\mathscr{X},\cdot)$. Formal application associates to the left. For each formal expression e, the set of free variables $FV(e)$ is defined as usual. A formal expression e is closed if the set $FV(e)$ is empty. We write $e[e'/x]$ for substitution on formal expressions. Closed formal expressions are interpreted in $\mathscr{X}$ by interpreting the formal application by the partial operation $\cdot\colon \mathscr{X} \times \mathscr{X} \rightharpoonup \mathscr{X}$.

We use the following notation. For a formal expression e with $FV(e) = \{x_1,\ldots,x_n\}$, we write $e\downarrow$ if, for all $r_1,\ldots,r_n \in \mathscr{X}$, $e[r_1/x_1]\ldots[r_n/x_n]$ is defined. For formal expressions e and e' with $FV(e) \cup FV(e') = \{x_1,\ldots,x_n\}$, we write $e \simeq e'$ if, for all $r_1,\ldots,r_n \in \mathscr{X}$, $e[r_1/x_1]\ldots[r_n/x_n]$ is defined if and only if $e'[r_1/x_1]\ldots[r_n/x_n]$ is defined and if both are defined then they denote the same element of $\mathscr{X}$.

Definition 3.4.2. A *partial combinatory algebra (PCA)* is a partial applicative structure $(\mathscr{X},\cdot)$ with distinguished elements $\mathbf{k},\mathbf{s} \in \mathscr{X}$ satisfying

$$\mathbf{k}\ x\ y\downarrow, \qquad \mathbf{k}\ x\ y \simeq x, \qquad \mathbf{s}\ x\ y\downarrow, \qquad \mathbf{s}\ x\ y\ z \simeq (x\ z)\ (y\ z).$$

Our intended example is the *Kleene PCA*. Its partial applicative structure is $(\mathbb{N},\cdot)$ having the natural numbers $\mathbb{N}$ as underlying set and Kleene application as the partial operation $\cdot$. The Kleene application $n \cdot m$ is defined as the n-th computable partial function in $\mathbb{N} \rightharpoonup \mathbb{N}$ applied to the number m. Further examples of partial combinatory algebras can be found in [61].

Proposition 3.4.3 (Combinatory Completeness)**.** *For each formal expression e and each variable x, there exists a formal expression $\lambda\!\!\lambda x.e$ with $FV(\lambda\!\!\lambda x.e) = FV(e) \setminus \{x\}$, $(\lambda\!\!\lambda x.e)\downarrow$ and $(\lambda\!\!\lambda x.e)\ x \simeq e$.*

Proof. See [61, Proposition 1.1.6]. □

Combinatory completeness means that we have $(\lambda\!\!\lambda x.e)\ r \simeq e[r/x]$ for all $r \in \mathscr{X}$. We should note that $\lambda\!\!\lambda x.e$ does *not* satisfy the general β-equation in which r is replaced by an arbitrary formal expression e'. Also, Kleene equality is not a congruence with respect to the meta-notation $\lambda\!\!\lambda x.e$, i.e. $e \simeq e'$ does *not* imply $\lambda\!\!\lambda x.e \simeq \lambda\!\!\lambda x.e'$. We refer the reader to [61, §1.1.2] for more information about these issues.

The next three propositions are almost exactly as in [62], where proofs can be found.

Proposition 3.4.4. *Each PCA $\mathscr{X}$ has elements pair, fst and snd satisfying*

$$\mathit{pair}\ x\ y\downarrow, \qquad \mathit{fst}\ (\mathit{pair}\ x\ y) \simeq x, \qquad \mathit{snd}\ (\mathit{pair}\ x\ y) \simeq y.$$

Proposition 3.4.5. *Each PCA $\mathscr{X}$ has, for each $n \in \mathbb{N}$, an element $\overline{n}$ as well as elements succ, pred, iszero satisfying*

$$\mathit{succ}\ \overline{n} \simeq \overline{n+1}, \qquad \mathit{pred}\ \overline{0} \simeq \overline{0}, \qquad \mathit{pred}\ \overline{n+1} \simeq \overline{n},$$

$$\mathit{iszero}\ \overline{0} \simeq \mathit{fst}, \qquad \mathit{iszero}\ \overline{n+1} \simeq \mathit{snd}.$$

Proposition 3.4.6. *Each PCA $\mathscr{X}$ has an element $\mathbf{z}$ satisfying $\mathbf{z}\ x\downarrow$ and $\mathbf{z}\ y\ x \simeq y\ (\mathbf{z}\ y)\ x$.*

Given the definition of numerals and recursion, it follows that a PCA is Turing complete. Making use of this fact, we will often describe algorithms informally when it is clear that they are computable, rather than giving specific codes for them.

Proposition 3.4.7. *Each PCA $\mathscr{X}$ has, for each finite list of elements $r_1,\dots,r_k \in \mathscr{X}$ with $0 \leq k \in \mathbb{N}$, an element $[r_1,\dots,r_k]$ as well as elements cons, hd, tl, isempty, map satisfying*

$$cons\ r\ [r_1,\dots,r_k] \simeq [r,r_1,\dots,r_k], \qquad hd\ [r_1,\dots,r_{k+1}] \simeq r_1, \qquad tl\ [r_1,r_2,\dots,r_k] \simeq [r_2,\dots,r_k],$$

$$map\ s\ [r_1,\dots,r_k] \simeq [s\ r_1,\dots,s\ r_k] \qquad isempty\ [] \simeq fst, \qquad isempty\ [r_1,\dots,r_{k+1}] \simeq snd.$$

Proof. Using natural numbers, pairing and Turing completeness. □

We now define the category of assemblies on sets with respect to a partial combinatory algebra.

Definition 3.4.8. An *assembly* A is a pair $(|A|, \|\cdot\|_A)$, where $|A|$ is a set and $\|\cdot\|_A$ is a function of type $|A| \to \mathscr{P}(\mathscr{X})$ assigning a non-empty set to each element $x \in |A|$.

Definition 3.4.9. A function $f\colon A \to B$ is *tracked* by a realizer $r \in \mathscr{X}$ if the following holds.

$$\forall x \in A.\ \forall a \in \|x\|_A.\ r \cdot a \downarrow\ \wedge\ r \cdot a \in \|f(x)\|_B$$

Definition 3.4.10. The category $\mathbf{Ass}_{\mathscr{X}}$ of *assemblies on* $\mathscr{X}$ has as objects the assemblies on $\mathscr{X}$ and the morphisms from A to B are all functions $f\colon |A| \to |B|$ that are tracked by some realizer $r \in \mathscr{X}$.

When clear from the context, we omit the explicit reference to the PCA $\mathscr{X}$, writing **Ass** for $\mathbf{Ass}_{\mathscr{X}}$.

It is well-known that $\mathbf{Ass}_{\mathscr{X}}$ is a quasi-topos. (It is equivalent to a category of separated objects in a topos, see [56, §6], and any such category is a quasi-topos, see [57, Theorem A4.4.5].) For reference, we recall the concrete construction of some of the structure of $\mathbf{Ass}_{\mathscr{X}}$, see [61] for more detail.

Terminal Object. A terminal object 1 may be defined by taking $|1| = 1$ and $\|\diamond\|_1 = \mathscr{X}$.

Pullbacks. The pullback of two maps $f\colon B \to A$ and $g\colon C \to A$ as in

$$\begin{array}{ccc} B \times_A C & \xrightarrow{\pi_2} & C \\ {\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle g} \\ B & \xrightarrow[f]{} & A \end{array}$$

can be constructed by letting $|B \times_A C| = \{\langle b,c\rangle \in |B| \times |C| \mid f(b) = g(c)\}$ and $\|\langle b,c\rangle\|_{B\times_A C} = \{pair\ r\ s \mid r \in \|b\|_B\ \wedge\ s \in \|c\|_C\}$. The maps π_1 and π_2 are the evident projections.

Initial Object. An initial object may be defined by taking $|0| = 0$ and letting the realizability predicate $\|\cdot\|_0$ be the unique function $\|\cdot\|_0\colon 0 \to \mathscr{P}(\mathscr{X})$.

Binary Coproducts. The binary coproduct $A+B$ may be defined by taking $|A+B| = |A|+|B|$, realized by $\|\kappa_1(x)\|_{A+B} = \{pair\ \overline{0}\ r \mid r \in \|x\|_A\}$ and $\|\kappa_2(y)\|_{A+B} = \{pair\ \overline{1}\ r \mid r \in \|y\|_B\}$.

Coequalisers. The coequaliser e as in the diagram below can be constructed by letting $|C| = |B|/_\sim$ and $\|c\|_C = \bigcup_{y\in c}\|y\|_B$, for the equivalence relation $\sim$ generated by $y \sim y' \iff \exists x \in |A|.\, f(x) = g(x)$.

$$A \underset{g}{\overset{f}{\rightrightarrows}} B \xrightarrow[e]{} C$$

Cartesian Closure. The exponent $(A \Rightarrow B)$ may be defined by letting $|A \Rightarrow B| = (|A| \Rightarrow |B|)$ and $\|f\|_{A\Rightarrow B} = \{r \in \mathscr{K} \mid r \text{ tracks } f\}$.

Weak Partial Map Classifiers. A monomorphism $m\colon A \rightarrowtail B$ is strong if m is isomorphic, as an object in the slice over B, to an object of the form $m_0\colon B_0 \rightarrowtail B$, where $|B_0| \subseteq |B|$ and $\|y_0\|_{B_0} = \|y_0\|_B$ for all $y_0 \in B_0$.

Weak partial map classifiers exist in **Ass** by Proposition 3.1.13. For each object A, a weak partial map classifier is given by defining $A_\perp$ by $|A_\perp| = |A|+1$, $\|\kappa_1(a)\|_{A_\perp} = \|\lambda x\colon 1.\, a\|_{(1\Rightarrow A)}$ and $\|\kappa_2(\diamond)\|_{A_\perp} = \mathscr{K}$, and defining $\eta_A\colon A \rightarrowtail A_\perp$ by $\eta_A(a) = \kappa_1(a)$. That η_A is strong follows because we have an isomorphism $(1 \Rightarrow A) \cong A$ in **Ass**. The universal property of η_A is given as follows. Let $m\colon B_0 \rightarrowtail B$ be a strong monomorphism. Without loss of generality, we can assume $|B_0| \subseteq |B|$, $\forall b \in |B_0|.\, \|b\|_{B_0} = \|b\|_B$ and that m is the canonical inclusion. Let $f\colon B_0 \to A$. Then the unique map $\tilde{f}$ making the square

$$\begin{array}{ccc} B_0 & \xrightarrow{f} & A \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta_A} \\ B & \xrightarrow[\tilde{f}]{} & A_\perp \end{array}$$

a pullback is defined by

$$\tilde{f}(b) = \begin{cases} \kappa_1(f(b)) & \text{if } b \in |B_0| \\ \kappa_2(\diamond) & \text{otherwise.} \end{cases}$$

This map is realized by the code $r = \lambda\!\!\lambda x.\, \lambda\!\!\lambda y.\, s\ x$ where s is a code tracking f. To see this, suppose $b \in |B_0|$ and $s' \in \|b\|_B$. Then $r\ s' = \lambda\!\!\lambda y.\, s\ s'$ and this code evidently realizes $\lambda x : 1.\, f(b)$. Thus, by definition of $\|\cdot\|_{A_\perp}$, it is a realizer of $\kappa_1(f(b)) \in |A_\perp|$. If $b \in (|B| \setminus |B_0|)$ and $s' \in \|b\|_B$ then $r\ s' = \lambda\!\!\lambda y.\, s\ s'$ and this code, like any code, realizes $\kappa_2(\diamond) \in |A_\perp|$. Notice that $\lambda\!\!\lambda y.\, s\ s'$ is defined even when $s\ s'$ is not, and this is the reason for using the set of realizers of $1 \Rightarrow A$ instead of A.

With this description of the weak partial map classifier, the exponent $(A \Rightarrow B_\perp)$ is isomorphic to the object $(A \rightharpoonup B)$ with underlying set $|A \rightharpoonup B| = |A \Rightarrow B_\perp|$ and

$$\|f\|_{A\rightharpoonup B} = \{r \in \mathscr{K} \mid \forall a \in |A|.\, \forall s \in \|a\|_A.\, (f(a) \in B) \implies (r\ s\downarrow \wedge\ r\ s \in \|f(a)\|_B\},$$

i.e. the realizers of f are the codes that track f where it is defined and that may have arbitrary behaviour where f is undefined.

Natural Number Object. An object of natural numbers is given by the pair $(\mathbb{N}, \|\cdot\|_{\mathbb{N}})$ in which $\mathbb{N}$ is the set of natural numbers and the realizers are defined by $\|n\|_{\mathbb{N}} = \{\overline{n}\}$, where $\overline{n}$ is as defined in Proposition 3.4.5. With the maps $z\colon 1 \to (\mathbb{N}, \|\cdot\|_{\mathbb{N}})$ and $s\colon (\mathbb{N}, \|\cdot\|_{\mathbb{N}}) \to (\mathbb{N}, \|\cdot\|_{\mathbb{N}})$ given by $z = 0$ and $s(n) = n+1$, this object becomes a natural number objects, since primitive recursion can be implemented using the fixed-point combinator **z**.

There is a forgetful functor $|\cdot|\colon \mathbf{Ass} \to \mathbf{Sets}$ that maps an assembly A to its underlying set and that is the identity on morphisms. It has a full and faithful right adjoint $\nabla\colon \mathbf{Sets} \to \mathbf{Ass}$ mapping a set B to the assembly $(|B|, \|\cdot\|_B)$ whose realizability predicate is defined by $\|x\|_B = \mathscr{X}$ for all $x \in B$.

$$\mathbf{Ass} \underset{|\cdot|}{\overset{\nabla}{\underset{\longrightarrow}{\overset{\longleftarrow}{\top}}}} \mathbf{Sets}$$

3.4.2 Permutation Actions on Assemblies

By analogy with the construction of the Schanuel topos as permutation actions in **Sets**, we construct the realizability category $\mathbf{Ass}_S(P)$ as a category of permutation actions in **Ass**.

We define a group object $(|P|, \|\cdot\|_P)$ of finite support permutations. The underlying set of P is the set of automorphisms on $\mathbb{N}$ with finite support.

$$|P| = \{p \in Aut(\mathbb{N}) \mid \exists A \in \mathscr{P}_{\text{fin}}(\mathbb{N}).\ \forall n \in (\mathbb{N} \setminus A).\ p(n) = n\}$$

The restriction to permutations with finite support makes it easy to define realizers for the permutations. As noted in Remark 3.3.9, this is no restriction, since the Schanuel topos may be defined using finite support permutations alone. For $p \in |P|$, the set $\|p\|_P$ of realizers for p can be taken to be the (finite) non-constant part of the graph of p.

$$\|p\|_P \stackrel{\text{def}}{=} \{[pair\ \overline{n_1}\ \overline{m_1}, \ldots, pair\ \overline{n_k}\ \overline{m_k}] \mid NCGraph(p) = \{\langle n_1, m_1\rangle, \ldots, \langle n_k, m_k\rangle\}\}$$

$$NCGraph(p) \stackrel{\text{def}}{=} \{\langle n, m\rangle \in \mathbb{N} \times \mathbb{N} \mid n \neq m \ \wedge\ p(n) = m\}$$

It is evident that, with this representation, the group operations $u\colon 1 \to |P|$, $m\colon |P| \times |P| \to |P|$ and $i\colon |P| \to |P|$ defined by $u = id$, $m(p,q) = p \circ q$ and $i(p) = p^{-1}$ respectively are all realizable, thus making P a group object in **Ass**.

The category P-**Ass** is a computable version of $|P|$-**Sets**. An object of P-**Ass** is a set $|A|$ with realizability information $\|\cdot\|_A$ together with a realizable action $\mu_A\colon |P| \times |A| \to |A|$. The underlying set functor $|\cdot|\colon \mathbf{Ass} \to \mathbf{Sets}$ can be lifted to a functor $|\cdot|\colon P\text{-}\mathbf{Ass} \to |P|\text{-}\mathbf{Sets}$ which maps $(A, \mu_A\colon P \times A \to A)$ to the underlying $|P|$-set $(|A|, |\mu_A|\colon |P| \times |A| \to |A|)$.

Proposition 3.4.11. *The underlying set functor* $|\cdot|\colon P\text{-}\mathbf{Ass} \to |P|\text{-}\mathbf{Sets}$ *preserves finite limits and has a full and faithful right adjoint.*

$$P\text{-}\mathbf{Ass} \underset{|\cdot|}{\overset{\nabla_P}{\underset{\longrightarrow}{\overset{\longleftarrow}{\top}}}} |P|\text{-}\mathbf{Sets}$$

Proof. The adjunction is defined using the adjunction

$$\mathbf{Ass} \underset{|\cdot|}{\overset{\nabla}{\underset{\longrightarrow}{\overset{\longleftarrow}{\top}}}} \mathbf{Sets}.$$

The right adjoint ∇_P maps an object $(A, \mu_A \colon |P| \times A \to A)$ to the object $(\nabla A, \mu_A \colon P \times \nabla A \to \nabla A)$. Note that μ_A can be viewed as both as a function $|P| \times A \to A$ in **Sets** and as a morphism $P \times \nabla A \to \nabla A$ in **Ass**. It is routine to verify that $|\cdot|$ preserves finite limits and that ∇_P is full and faithful. □

As in the definition of Schanuel topos, we are interested only in objects with the finite support property. We therefore restrict to those objects in P-**Ass** whose underlying $|P|$-set has the finite support property, i.e. is an object of the Schanuel topos.

Definition 3.4.12. The category $\mathbf{Ass}(P)$ is the full subcategory of P-**Ass** defined by taking all those objects A for which $|A|$ has the finite support property.

The category $\mathbf{Ass}(P)$ relates to the Schanuel topos $\mathbb{S}$ in the same way as the category of assemblies **Ass** relates to the category of sets. The underlying set functor $|\cdot| \colon P\text{-}\mathbf{Ass} \to |P|\text{-}\mathbf{Sets}$ restricts to a functor $|\cdot| \colon \mathbf{Ass}(P) \to \mathbb{S}$ and we have:

Proposition 3.4.13. *The underlying set functor $|\cdot| \colon \mathbf{Ass}(P) \to \mathbb{S}$ preserves finite limits and has a full and faithful right adjoint.*

$$\mathbf{Ass}(P) \underset{|\cdot|}{\overset{\nabla_P}{\underset{\longrightarrow}{\overset{\longleftarrow}{\top}}}} \mathbb{S}$$

Proposition 3.4.14. *The following square is a pullback of categories.*

$$\begin{array}{ccc} \mathbf{Ass}(P) & \xrightarrow{|\cdot|} & \mathbb{S} \\ {\scriptstyle I}\downarrow & & \downarrow{\scriptstyle I} \\ P\text{-}\mathbf{Ass} & \xrightarrow[|\cdot|]{} & |P|\text{-}\mathbf{Sets} \end{array}$$

The point of this pullback is that it gives us two ways of viewing the objects of $\mathbf{Ass}(P)$. First, by definition of $\mathbf{Ass}(P)$, an object is a triple $((|A|, \|\cdot\|_A), \mu)$ where $(|A|, \|\cdot\|_A)$ is an object of **Ass** and μ is a morphism of type $P \times (|A|, \|\cdot\|_A) \to (|A|, \|\cdot\|_A)$ in **Ass**, such that $(|A|, |\mu| \colon |P| \times |A| \to |A|)$ is an object in $\mathbb{S}$. Because of the above pullback, we can also view the objects of $\mathbf{Ass}(P)$ as pairs $(|A|, \|\cdot\|_{UI|A|})$ where $|A|$ is a Schanuel set $(UI|A|, \mu_{UI|A|})$ and $(UI|A|, \|\cdot\|_{UI|A|})$ is an object in **Ass** for which $\mu_{UI|A|}$ is realizable. We can therefore view the objects of $\mathbf{Ass}(P)$ as Schanuel sets that are equipped with realizability information such that the permutation action is realizable. We will often use this second view and leave implicit applications the forgetful functors, writing just $(|A|, \|\cdot\|_A)$ for the object $(UI|A|, \|\cdot\|_{UI|A|})$ of $\mathbf{Ass}(P)$.

We have an analogue of Proposition 3.3.7 for $\mathbf{Ass}(P)$.

Proposition 3.4.15. *The full and faithful inclusion functor* $I\colon \mathbf{Ass}(P) \to P\text{-}\mathbf{Ass}$ *preserves finite limits and has a right adjoint*

$$\mathbf{Ass}(P) \underset{I}{\overset{\rho}{\underset{\longrightarrow}{\overset{\longleftarrow}{\top}}}} P\text{-}\mathbf{Ass}\,.$$

Proof. The adjunction is defined as in Proposition 3.3.7. It is straightforward to verify that all maps involved have realizers. □

The situation may be summarised by the following (non-commuting) diagram.

$$\begin{array}{ccc}
\mathbf{Ass}(P) & \overset{\longleftarrow}{\top}\ \longrightarrow & \mathbb{S} \\
\downarrow\!\dashv\uparrow & & \downarrow\!\dashv\uparrow \\
P\text{-}\mathbf{Ass} & \overset{\longleftarrow}{\top}\ \longrightarrow & |P|\text{-}\mathbf{Sets} \\
\uparrow\!\dashv\downarrow\!\dashv\uparrow & & \uparrow\!\dashv\downarrow\!\dashv\uparrow \\
\mathbf{Ass} & \overset{\longleftarrow}{\top}\ \longrightarrow & \mathbf{Sets}
\end{array} \tag{3.16}$$

The right-hand column is the essentially the situation (3.3) for the Schanuel topos, only with finite-support permutations, see also [39]. In the left column, P-**Ass** is both monadic and comonadic over **Ass** and $\mathbf{Ass}(P)$ is comonadic over P-**Ass**.

Proposition 3.4.16. *The category* $\mathbf{Ass}(P)$ *is a quasi-topos.*

Proof. This follows from Proposition 3.1.14 since **Ass** is a quasi-topos and $\mathbf{Ass}(P)$ is comonadic over P-**Ass**, which in turn is comonadic over **Ass**, each with a finite limit preserving comonad. □

We may use the left-hand column in the above situation to calculate the structure of $\mathbf{Ass}(P)$. We give a few constructions to illustrate the view of $\mathbf{Ass}(P)$ as a computable version of $\mathbb{S}$.

Terminal Object. A terminal object in $\mathbf{Ass}(P)$ is given by $(1, \|\cdot\|_1)$, where 1 is the terminal object in $\mathbb{S}$ and $\|\diamond\|_1 = \mathscr{X}$. With this definition, the action on 1 is trivially computable.

Pullbacks. The pullback of two maps $f\colon B \to A$ and $g\colon C \to A$ as in

$$\begin{array}{ccc}
B \times_A C & \xrightarrow{\pi_2} & C \\
{\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle g} \\
B & \xrightarrow[f]{} & A
\end{array}$$

can be defined by letting the underlying set $|B \times_A C|$ be $\{\langle b,c\rangle \in |B| \times |C| \mid f(b) = g(c)\}$ with action $\pi \cdot_{|B\times_A C|} \langle b,c\rangle = \langle \pi \cdot_{|B|} b, \pi \cdot_{|C|} c\rangle$ and with realizability predicate $\|\langle b,c\rangle\|_{B\times_A C} = \{pair\ r\ s \mid r \in \|b\|_B \wedge s \in \|c\|_C\}$. The action on $B \times_A C$ is realizable because the actions on B and C are. If r_1 and r_2 realize the action on B and C, then $\lambda p.\lambda x.pair\ (r_1\ p\ (fst\ x))\ (r_2\ p\ (snd\ x))$ is a realizer making the action $\cdot_{|B\times_A C|}$ a map of type $P \times (B \times_A C) \to (B \times_A C)$ in $\mathbf{Ass}(P)$. The maps π_1 and π_2 are the evident projections.

Cartesian Closed Structure. The exponent $(A, \|\cdot\|_A) \Rightarrow (B, \|\cdot\|_B)$ in $\mathbf{Ass}(P)$ can be defined to be $(A \Rightarrow B, \|\cdot\|_{A\Rightarrow B})$ where $A \Rightarrow B$ is the exponent in $\mathbb{S}$ and $\|f\|_{A\Rightarrow B} = \{r \in \mathscr{X} \mid r \text{ tracks } f\}$. Recall that the exponent $A \Rightarrow B$ in $\mathbb{S}$ consists of all functions from A to B having finite support with respect to the action $\cdot_{A\Rightarrow B}$ defined by $\pi \cdot_{A\Rightarrow B} f = \lambda x.\, \pi \cdot_B (f(\pi^{-1} \cdot_A x))$. Notice that $\cdot_{A\Rightarrow B}$ is realizable because all of f, $\cdot_A$, $\cdot_B$ and $i\colon P \to P$ are realizable.

Natural Number Object. The natural number object $\mathbb{N}$ is defined as $(\mathbb{N}_{\mathbf{Ass}}, \cdot_{\mathbb{N}})$ where $\mathbb{N}_{\mathbf{Ass}}$ is the natural number object in **Ass**, and $\cdot_{\mathbb{N}}\colon P \times \mathbb{N} \to \mathbb{N}$ is the trivial action given by $\pi \cdot_{\mathbb{N}} n = n$.

Object of Names. The object of names $\mathbf{N}$ differs from the natural numbers $\mathbb{N}$ only in the definition of its action. The object $\mathbf{N}$ is defined as $(\mathbb{N}_{\mathbf{Ass}}, \cdot_{\mathbf{N}})$ where $\mathbb{N}_{\mathbf{Ass}}$ is the natural number object in **Ass** and the action is given by $\pi \cdot_{\mathbf{N}} n = \pi(n)$. By definition of the group object P, this action is evidently realizable.

Weak Partial Map Classifiers. Strong monomorphisms have the same description as in **Ass**. Weak partial map classifiers, too, have the same description as in **Ass** with the evident action. For each object A, the weak partial map classifier $\eta\colon A \to A_\perp$ from **Ass** becomes a classifier in $\mathbf{Ass}(P)$ when $A_\perp$ is equipped with the action $\pi \cdot_{A_\perp} \kappa_1(a) = \kappa_1(\pi \cdot_A a)$ and $\pi \cdot_{A_\perp} \kappa_2(\diamond) = \kappa_2(\diamond)$. Recalling the definitions $\|\kappa_1(a)\|_{A_\perp} = \|\lambda x : 1.\, a\|_{(1\Rightarrow A)}$ and $\|\kappa_2(\diamond)\|_{A_\perp} = \mathscr{X}$, this action $\cdot_{A_\perp}\colon P \times A_\perp \to A_\perp$ is realized by the code $\lambda\!\!\lambda p.\, \lambda\!\!\lambda a.\, \lambda\!\!\lambda u.\, r_A\ p\ (a\ u)$, where r_A is a realizer for the action $\cdot_A\colon P \times A \to A$.

As a special case, a weak subobject classifier is given by the map $\kappa_1\colon 1 \to \nabla_P 2$.

Monoidal Structure. We define a strict affine symmetric monoidal structure $*\colon \mathbf{Ass}(P) \times \mathbf{Ass}(P) \to \mathbf{Ass}(P)$ by letting $(A, \|\cdot\|_A) * (B, \|\cdot\|_B)$ be the object $(A * B, \|\cdot\|_{A*B})$ where $A * B$ is the monoidal product in $\mathbb{S}$ with $A * B = \{\langle a, b\rangle \in A \times B \mid \mathrm{supp}(a) \cap \mathrm{supp}(b) = \emptyset\}$ and $\pi \cdot_{A*B} \langle a, b\rangle = \langle \pi \cdot_A a, \pi \cdot_B b\rangle$ and $\|\langle a, b\rangle\|_{A*B} = \{pair\ r\ s \mid r \in \|a\|_A \ \wedge\ s \in \|b\|_B\}$.

For this definition of the monoidal structure, it is immediate that the monomorphism $A * B \rightarrowtail A \times B$ is strong for all A and B. Hence property (C1) of Section 3.2 holds. Since the underlying set $A * B$ is defined as in $\mathbb{S}$, it is readily seen that property (C2) is inherited from $\mathbb{S}$. However, property (C3), that $- * A$ preserves exact sequences, does not appear to transfer directly from the Schanuel topos. In the proof of this property for the Schanuel topos (Lemma 3.3.13) we have made use of the fact that we can always 'freshen' certain names in all elements without affecting any other names. In the next section we see that such freshening is not available in $\mathbf{Ass}(P)$. This will lead us to the definition of the subcategory $\mathbf{Ass}_S(P)$ of $\mathbf{Ass}(P)$ in which freshening is available.

3.4.3 Support Approximations

The main ingredient missing from the category $\mathbf{Ass}(P)$ is the possibility to choose fresh names effectively. We have restricted the objects of $\mathbf{Ass}(P)$ so that all elements have finite support, with the intention that this makes it always possible to find fresh names. While this restriction allows us to always find

fresh names externally, it is not sufficient to do so internally. Let A be an object of $\mathbf{Ass}(P)$ and suppose we have an element $a \in |A|$ for which we want to compute a fresh name. By this we mean that we want a realizer $c \in \mathscr{X}$ such that, for any realizer $r \in \|a\|$, the application $c\ r$ is defined and is a realizer of some name n such that $(pair\ r\ (c\ r))$ realizes $\langle a,n\rangle \in A * \mathbf{N}$. Note that this cannot be expressed as a morphism of $\mathbf{Ass}(P)$, since the function that is realized by c is not equivariant and therefore not a morphism in $\mathbf{Ass}(P)$. A code such as c is useful to realize constructions like $(\text{new } n.M)$, which amounts first choosing a fresh name n and then computing M using that fresh name. In general, $\mathbf{Ass}(P)$ does not have such a code c, as the following example shows.

Example 3.4.17. We show that in general no code c with the above property exists. We show this for the case where the underlying PCA of $\mathbf{Ass}(P)$ is the Kleene PCA. Consider the object $(\mathbf{N} \Rightarrow \mathbf{N})$ of $\mathbf{Ass}(P)$. A realizer for an element $f \in |\mathbf{N} \Rightarrow \mathbf{N}|$ is a natural number representing a total function $\mathbb{N} \Rightarrow \mathbb{N}$ in **Sets**. We show that if there is a code c generating fresh names for this type then we can solve the special halting problem, i.e. we can decide whether or not a Turing Machine encoded by a natural number K halts on input K.

For any code K for a Turing Machine, we define the following total computable function in $\mathbb{N} \Rightarrow \mathbb{N}$.

$$f_K(n) = \begin{cases} 0 & \text{if } n > 0 \text{ and the Turing Machine } K \text{ stops with input } K \text{ in} \leq n \text{ steps} \\ n & \text{otherwise} \end{cases}$$

Note that f_K can be obtained effectively, i.e. the function mapping K to a code for f_K is computable.

In order for f_K to denote an element of $\mathbf{N} \Rightarrow \mathbf{N}$, it must have finite support with respect to the action on $\mathbf{N} \Rightarrow \mathbf{N}$. In the case where the Turing Machine K does not stop, f_K is the identity and therefore satisfies $\text{supp}(f_K) = \emptyset$. In the case where the Turing Machine K stops after exactly k steps, one can show $\text{supp}(f_K) = \{0,1,\ldots,k\}$.

Now, suppose we have a code c that, given a realizer for f_K, computes a natural number n with the property $n \notin \text{supp}(f_K)$. Since f_K is total and tracked by some code, we can then compute the value $f_K(n)$. We show that $f_K(n) = n$ holds if and only if the Turing Machine K does not stop. Suppose $f_K(n) = n$. If $n = 0$ then the Turing Machine K does not stop, because otherwise we would have $0 \in \text{supp}(f_K)$, and we have assumed $0 \notin \text{supp}(f_K)$. If $n \neq 0$ then $f_K(n) = n$ implies that the Turing Machine K does not stop after n steps. We use $n \notin \text{supp}(f_K)$ to show that K never stops. Suppose K stops, say after exactly k steps. Since K has not stopped after n steps we have $k > n$. Then, by construction of f_K, we have $\text{supp}(f_K) = \{0,\ldots,k\}$. But since $n \notin \text{supp}(f_K)$ this implies $n > k$, and this leads to the contradiction $k > n > k$. Hence, if $f_K(n) = n$ then the Turing Machine K does not stop. Now suppose $f_K(n) \neq n$. This can only be the case if $n \neq 0$ and $f_K(n) = 0$. But in this case, K stops after no more than n steps.

We have thus shown that c can be used to decide the special halting problem. Since this problem is undecidable, no such code c can exist.

□

A consequence of the failure of $\mathbf{Ass}(P)$ to have a code for choosing fresh names is that $\mathbf{Ass}(P)$ does

not have interesting Σ^*-types. In fact, in $\mathbf{Ass}(P)$ even $W_{\mathbf{N}}$ does not have a fibred left adjoint $\Sigma^*_{\mathbf{N}}$. The reason for this failure is that $\Sigma^*_{\mathbf{N}}$-types can be used to implement unique choice of fresh names, as we shall see in Chapter 11. Another consequence of the failure of $\mathbf{Ass}(P)$ to have a code for choosing fresh names is that we cannot construct Π^*-types. It is not clear how to show that the functor $- * \mathbf{N}$ preserves exact sequences, as it should if it were to be a pullback-preserving left adjoint. In contrast to Σ^*-types, however, we do not have a proof for the non-existence of Π^*-types.

We view the non-existence of a code for choosing fresh names as a defect of particular choices of the sets of realizers $\|\cdot\|_A$. In the above example, the object $\mathbf{N} \Rightarrow \mathbf{N}$ is realized by codes for total computable function on the natural numbers. Given only such a code, one cannot expect to be able to compute the support of the function $f \in |\mathbf{N} \Rightarrow \mathbf{N}|$ it represents. A more reasonable encoding for f would be a pair $(\mathit{pair}\ s\ r)$ where s is a *support approximation* encoding a finite set that contains the support of f and r is a realizer of the function f. For such an encoding, choosing a fresh name is trivial: We can simply return any number not in the finite set encoded by s.

To define support approximations formally, we consider the Schanuel set of finite name-sets and equip it with realizability information to make it an object of $\mathbf{Ass}(P)$. The object $\mathscr{P}_{\mathrm{fin}}(\mathbf{N})$ consists of all finite sets of names. Recall from the discussion after Proposition 3.4.14 that an object of $\mathbf{Ass}(P)$ amounts to an object of $\mathbb{S}$ with a realizability predicate $\|\cdot\|$ on its underlying set such that the permutation action is tracked by some code. We define the realizability predicate $\|\cdot\|_{\mathscr{P}_{\mathrm{fin}}(\mathbf{N})}$ by encoding the finite name-sets as finite lists

$$\|A\|_{\mathscr{P}_{\mathrm{fin}}(\mathbf{N})} = \{[\overline{n_1}, \ldots, \overline{n_k}] \mid \text{if } A \text{ is } \{n_1, \ldots, n_k\}\}.$$

The action on $\mathscr{P}_{\mathrm{fin}}(\mathbf{N})$ is tracked by $(\mathit{map}\ r)$, where r is a code tracking the action $\cdot_{\mathbf{N}} \colon P \times \mathbf{N} \to \mathbf{N}$. The object $\mathscr{P}(\mathscr{P}_{\mathrm{fin}}(\mathbf{N}))$ in $\mathbb{S}$ consists of all sets of finite sets of names. We make it an object in $\mathbf{Ass}(P)$ by equipping it with the realizability predicate

$$\|S\|_{\mathscr{P}(\mathscr{P}_{\mathrm{fin}}(\mathbf{N}))} = \bigcup_{A \in S} \|A\|_{\mathscr{P}_{\mathrm{fin}}(\mathbf{N})}.$$

Note that the definitions of both the objects $\mathscr{P}_{\mathrm{fin}}(\mathbf{N})$ and $\mathscr{P}(\mathscr{P}_{\mathrm{fin}}(\mathbf{N}))$ in $\mathbf{Ass}(P)$ are not what one would get using the weak subobject classifier. We do not use the weak power object $A \Rightarrow \Omega$ as it does not have the right computability information in $\mathbf{Ass}(P)$. For example, the code for the identity $\lambda x.x$ realizes all elements of $A \Rightarrow \Omega$.

Definition 3.4.18. An object A in $\mathbf{Ass}(P)$ has *support approximations* if the function $f \colon A \to \mathscr{P}(\mathscr{P}_{\mathrm{fin}}(\mathbf{N}))$ defined by $f(x) = \{S \in \mathscr{P}_{\mathrm{fin}}(\mathbf{N}) \mid S \supseteq \mathrm{supp}(x)\}$ is tracked by some realizer.

Definition 3.4.19. An object A in $\mathbf{Ass}(P)$ has *computable support* if the function $\mathrm{supp} \colon A \to \mathscr{P}_{\mathrm{fin}}(\mathbf{N})$ is tracked by some realizer.

3.4.4 The Category $\mathbf{Ass}_S(P)$

Definition 3.4.20. The category $\mathbf{Ass}_S(P)$ is the full subcategory of $\mathbf{Ass}(P)$ consisting of all objects that have support approximations.

Proposition 3.4.21. *The full and faithful inclusion* $I\colon \mathbf{Ass}_S(P) \hookrightarrow \mathbf{Ass}(P)$ *preserves finite limits and has a right adjoint.*

$$\mathbf{Ass}_S(P) \underset{I}{\overset{S}{\underset{\hookrightarrow}{\overset{\leftarrow}{\top}}}} \mathbf{Ass}(P)\,.$$

Proof. The right adjoint S maps an object A to the object $((|B|, \|\cdot\|_B), \mu_B)$ where $|B| = |A|$, $\mu_B = \mu_A$ and

$$\|a\|_B = \{pair\ s\ r \mid X \in |\mathscr{P}_{\mathrm{fin}}(\mathbf{N})| \wedge X \supseteq \mathrm{supp}(a) \wedge s \in \|X\|_{\mathscr{P}_{\mathrm{fin}}(\mathbf{N})} \wedge r \in \|a\|_A\}\,.$$

The counit $\varepsilon_A\colon ISA \to A$ is given by the identity function $|ISA| = |A| \to |A|$ and is realized by *snd*.

We show preservation of finite limits as in the proof of Proposition 3.3.7. As shown there, it suffices to show that, for all objects A and B in $\mathbf{Ass}_S(P)$ and all monomorphisms $E \rightarrowtail IA$ in $\mathbf{Ass}(P)$, the morphisms $\varepsilon_1\colon IS1 \to 1$ and $\varepsilon_{IA\times IB}\colon IS(IA\times IB) \to (IA\times IB)$ and $\varepsilon_E\colon ISE \to E$ are isomorphisms. It is easily seen that to find inverses of these morphisms it suffices to show that their codomains have support approximations. We only consider the case for ε_E. Since there is a monomorphism $m\colon E \rightarrowtail IA$, since IA has support approximations, and since $\mathrm{supp}(e) = \mathrm{supp}(m(e))$ holds for all $e \in |E|$, we get a support approximation for E by first mapping along m and then using the support approximation for IA. □

Corollary 3.4.22. *The category* $\mathbf{Ass}_S(P)$ *is a quasi-topos.*

Proof. Using Propositions 3.4.21 and 3.1.14 with the fact that $\mathbf{Ass}(P)$ is a quasi-topos. □

Although in general we can only approximate the support in $\mathbf{Ass}_S(P)$, there are classes of objects on whose elements the support can be computed exactly.

Definition 3.4.23. An object A in $\mathbf{Ass}_S(P)$ has *decidable equality* if there is a morphism $e\colon A\times A \to 1+1$ in $\mathbf{Ass}_S(P)$ such that $e(a,a') = \kappa_1(\diamond)$ holds if and only if $a = a'$ holds.

Proposition 3.4.24. *Every object* A *in* $\mathbf{Ass}_S(P)$ *with decidable equality has computable support.*

Proof. Let $a \in A$. Since $\mathbf{Ass}_S(P)$ has support approximations, we can readily compute a finite set containing the support of a. Moreover, we have $n \in \mathrm{supp}(a) \iff (\exists n'.\ n' \# \langle n,a\rangle \wedge (n'\ n)\cdot a \neq a)$ and $n \notin \mathrm{supp}(a) \iff (\exists n'.\ n' \# \langle n,a\rangle \wedge (n'\ n)\cdot a = a)$. Since the permutation is computable, equality is decidable, and since we can generate a fresh name n' by choosing an arbitrary name not in the support approximation of $\langle n,a\rangle$, we can therefore use this property to test which names in the (finite) support approximation of a belong to the support of a. Hence, A has computable support. □

We spell out some constructions in $\mathbf{Ass}_S(P)$.

Finite Limits and Colimits. Finite limits and colimits can be constructed exactly as in $\mathbf{Ass}(P)$, since finite limits and colimits always have support approximations. For example, because of the inclusion $\mathrm{supp}(\langle a,b\rangle) \subseteq \mathrm{supp}(a) \cup \mathrm{supp}(b)$, a support approximation for $\langle a,b\rangle \in A \times B$ is given by the union of the support approximations for a and b.

Cartesian Closed Structure. Given Proposition 3.4.21, the exponent $A \Rightarrow B$ of two objects A and B in $\mathbf{Ass}_S(P)$ can be constructed as $S(IA \Rightarrow IB)$ using the exponent in $\mathbf{Ass}(P)$. This follows from the coreflection $I \dashv S$ in the same way as the construction of exponents in the Schanuel topos.

Monoidal Structure. The monoidal structure $*$ on $\mathbf{Ass}_S(P)$ is defined exactly as in $\mathbf{Ass}(P)$.

Proposition 3.4.25. *Binary coproducts in* $\mathbf{Ass}_S(P)$ *are disjoint and universal.*

Proof. This follows in the same way as for **Ass**, see e.g. [13, Theorem 3.6.23] for a proof. □

Example 3.4.26. We reconsider Example 3.4.17, this time in $\mathbf{Ass}_S(P)$. The exponent $\mathbf{N} \Rightarrow \mathbf{N}$ in $\mathbf{Ass}_S(P)$ is given by $S(I\mathbf{N} \Rightarrow I\mathbf{N})$, which means that it has the same underlying set as in $\mathbf{Ass}(P)$, but the realizers of an element $f \in (\mathbf{N} \Rightarrow \mathbf{N})$ are now pairs $(pair\ s\ r)$ where $s \in \mathscr{X}$ encodes a finite set containing the support of f and $r \in \mathscr{X}$ is a code tracking f. To give a code that computes a fresh name for an arbitrary element of $\mathbf{N} \Rightarrow \mathbf{N}$, it is enough to give a code that returns (a code for) the smallest natural number not in the finite set encoded by s. This is of course possible, so that in $\mathbf{Ass}_S(P)$ the previous undecidability result of Example 3.4.17 does not apply. The point where the proof in Example 3.4.17 goes wrong in $\mathbf{Ass}_S(P)$ is where we need to effectively compute a realizer for f_K for a given K encoding a Turing machine. In order to compute a realizer for f_K, we need to compute a support approximation for it and this amounts to giving an upper bound for the number of computation steps of the Turing machine K if the Turing machine K stops. This is not possible, as we do not know whether K will stop at all. □

Example 3.4.27. In $\mathbf{Ass}_S(P)$ the object $\mathbf{N} \Rightarrow A$ has decidable equality whenever A has decidable equality. The algorithm deciding whether two functions f and g in $\mathbf{N} \Rightarrow A$ are equal rests on the following simple observation. Let X be a finite set of names satisfying $\mathrm{supp}(f) \cup \mathrm{supp}(g) \subseteq X$. Let n be a name not in X. Then, $f = g$ holds if and only if the following property holds.

$$(\forall m \in X.\ f(m) = g(m)) \wedge (f(n) = g(n)) \tag{3.17}$$

To see this, let n' be any name. If $n' \in X$ then $f(n') = g(n')$ is immediate. Otherwise, if $n' \notin X$, then $(n\ n') \cdot f = f$ and $(n\ n') \cdot g = g$ hold, as both n and n' are fresh for f and g. This implies

$$f(n') = ((n\ n') \cdot f)(n') = ((n\ n') \cdot f)((n\ n') \cdot n) = (n\ n') \cdot (f(n)) = (n\ n') \cdot (g(n)) = g(n').$$

In $\mathbf{Ass}_S(P)$, we can compute a finite set X as above for arbitrary codes for f and g by computing the union of the support approximations for f and g. Moreover, we can compute a name n not in X. Since the objects $\mathbf{N}$ and A have decidable equality and since X is finite, the property (3.17) can then be checked effectively. Thus, $\mathbf{N} \Rightarrow A$ has decidable equality.

This situation in $\mathbf{Ass}_S(P)$ is in contrast to the situation in $\mathbf{Ass}(P)$ where the object $\mathbf{N} \Rightarrow A$ (note that this exponent is constructed differently) does not in general have decidable equality. For example, the object $\mathbf{N} \Rightarrow \mathbf{N}$ cannot have decidable equality, since otherwise we could solve the halting problem by comparing the function f_K in Example 3.4.17 to the identity function. □

3.4.4.1 Simple Monoidal Products

Lemma 3.4.28. *For each object A in $\mathbf{Ass}_S(P)$ having computable support, the functor $- * A \colon \mathbb{S} \to \mathbb{S}$ preserves exact sequences.*

Proof. It suffices to show that all the maps in Proposition 3.3.13 have realizers. By the construction of pullbacks and coequalisers, this is straightforward for all maps except $i \colon (B/R) * A \to (B * A)/(R * A)$. The morphism i is defined as taking an element $\langle [b]_R, a \rangle \in (B/R) * A$, computing a permutation π that exchanges all the names in $\mathrm{supp}(a)$ with fresh names and leaves all other names unchanged, and returning $[\pi \cdot b, a]_{R*A}$. By construction of coequalisers, we have $\|c\|_{(B*A/R*A)} = \bigcup_{x \in c} \|x\|_{B*A}$ for all $c \in (B * A/R * A)$. Therefore, to realize i it suffices to find a code r that realizes the mapping $\langle b, a \rangle \mapsto \langle \pi \cdot b, a \rangle$. For this, it suffices to find a code s that, given codes for $b \in B$ and $a \in A$, returns a code for a permutation $\pi \in P$ that exchanges all names in a for fresh names and that does not change the names in b. Since A has computable support, we can compute the support of a. Since B has support approximations, we can find $|\mathrm{supp}(a)|$ names that are fresh for both a and b. Using these two finite sets, it is straightforward using Turing-completeness to construct a code for a permutation $\pi \in P$ satisfying the above requirements. □

Proposition 3.4.29. *For each object A in $\mathbf{Ass}_S(P)$ having computable support, the functor $- * A \colon \mathbb{S} \to \mathbb{S}$ has a right adjoint.*

Proof. We use Proposition 3.2.5. Of the conditions (C1), (C2) and (C3) required for this proposition, (C1) and (C2) are as in the case for the Schanuel topos, and (C3) is given by Lemma 3.4.28 above. □

As observed in Proposition 2.5.6, the monoidal closed structure $(*, -*)$ provides us with simple monoidal products Π^* in the codomain fibration $cod \colon \mathbf{Ass}_S(P)^{\to} \to \mathbf{Ass}_S(P)$.

3.4.4.2 Simple Monoidal Sums

In this section we construct simple monoidal sums in $\mathbf{Ass}_S(P)$. The functor Σ_A^* as in the diagram below has on the underlying sets the same definition as in the Schanuel topos.

$$\begin{array}{ccc} \mathbf{Ass}_S(P)/(- * A) & \xrightarrow{\Sigma_A^*} & \mathbf{Ass}_S(P)^{\to} \\ & {\scriptstyle Gl\,(-*A)} \searrow \quad \swarrow {\scriptstyle cod} & \\ & \mathbf{Ass}_S(P) & \end{array} \tag{3.18}$$

An object $f\colon B \to \Gamma * A$ is mapped to the object $\Sigma_A^* f\colon \Sigma_A^* B \to \Gamma$ defined by $\Sigma_A^* B = \{[b]_f \mid b \in B\}$, $\pi \cdot [b]_f = [\pi \cdot b]_f$ and $(\Sigma_A^* f)([b]_f) = \dot{\pi}_1(f(b))$. This definition of Σ_A^* is equipped with realizers in a way which matches the usual definition of quotient types in realizability categories.

$$\|c\|_{\Sigma_A^* B} = \bigcup_{b \in c} \|b\|_B$$

The action on $\Sigma_A^* B$ is realizable since it is given pointwise and since B, being an object of $\mathbf{Ass}_S(P)$, has a realizable action. The morphism $\Sigma_A^* f$ in $\mathbf{Ass}_S(P)$ is realizable since $\dot{\pi}_1$ and f are. Furthermore, $\Sigma_A^* B$ has support approximations since, because of $\mathrm{supp}([b]_f) \subseteq \mathrm{supp}(b)$, we can simply use the support approximations of B. The morphism part of Σ_A^* is defined exactly as in Section 3.3.2. Since the morphism action of Σ_A^* given there is defined pointwise, it is easily seen that the morphisms are realizable.

With this definition we can ask the question when Σ_A^* is a fibred functor. The proof of the following lemma makes essential use of support approximations.

Lemma 3.4.30. *If A is an object of $\mathbf{Ass}_S(P)$ and A has computable support, then Σ_A^* is a fibred functor from $Gl(-*A)$ to cod.*

Proof. We need to show that if the square on the left is a pullback then so is the square on the right.

$$\begin{array}{ccc} B & \xrightarrow{u} & C \\ {\scriptstyle f}\downarrow & \lrcorner & \downarrow{\scriptstyle g} \\ \Gamma * A & \xrightarrow[v*A]{} & \Delta * A \end{array} \qquad \begin{array}{ccc} \Sigma_A^* B & \xrightarrow{\Sigma_A^* u} & \Sigma_A^* C \\ {\scriptstyle \Sigma_A^* f}\downarrow & & \downarrow{\scriptstyle \Sigma_A^* g} \\ \Gamma & \xrightarrow[v]{} & \Delta \end{array}$$

The proof goes essentially as the proof of Lemma 3.3.31. However, the existence statement in that lemma now involves effectivity. We have to give an isomorphism $i\colon \Sigma_A^* B \to \Gamma \times_\Delta \Sigma_A^* C$ satisfying $\Sigma_A^* f = \pi_1 \circ i$ and $\Sigma_A^* u = \pi_2 \circ i$, where we use the same notation as in the construction of pullbacks for $\mathbf{Ass}(P)$. We define $i([b]_f) = \langle \dot{\pi}_1(f(b)), [u(b)]_g \rangle$. This map is evidently realizable and satisfies the two required equations. Now we consider the inverse of i. By examining the proof of Lemma 3.3.31, we can informally describe i^{-1} as follows. Given $\langle \gamma, [c]_g \rangle \in \Gamma \times_\Delta \Sigma_A^* C$, find a permutation π such that $[c]_g = [\pi \cdot c]_g$ holds and $\dot{\pi}_2(g(\pi \cdot c))$ is fresh for γ, use the left pullback for $\langle \gamma, \dot{\pi}_2(g(\pi \cdot c)) \rangle \in \Gamma * A$ and $\pi \cdot c$ to obtain $b \in B$, and return $[b]_f$ as the result. Note that we need to freshen c in the definition in order to make use of the left pullback, since $\langle \gamma, \dot{\pi}_2(g(c)) \rangle$ need not be an element of $\Gamma * A$. There is only one step in this description of i^{-1} for which it is not trivial to find a realizer, namely to find a permutation π such that $[c]_g = [\pi \cdot c]_g$ holds and $\dot{\pi}_2(g(\pi \cdot c))$ is fresh for γ. If we can compute the support of $\dot{\pi}_2(g(c)) \in A$ and we can finitely approximate the support of $\langle \gamma, c \rangle$ then we can effectively compute such a π as follows. First, compute the support of $\dot{\pi}_2(g(c))$ and an approximation of the support of $\langle \gamma, c \rangle$. Compute the first $|\mathrm{supp}(\dot{\pi}_2(g(c)))|$ names that are not in $\mathrm{supp}(\gamma, c)$. This always succeeds because approximations are finite. Hence, we return (the finite graph of) a permutation that swaps $\mathrm{supp}(\dot{\pi}_2(g(c)))$ for these fresh names and leaves all other names unchanged. Under the assumptions of the proposition, we can therefore realize i^{-1}, making it a morphism of $\mathbf{Ass}_S(P)$. Finally, that the morphisms i and i^{-1}

define an isomorphism then follows as in Lemma 3.3.31. Notice that in this argument it is necessary to compute the support of $\dot{\pi}_2(g(c))$ exactly, since we have $[c]_g = [\pi \cdot c]_g$ only if π fixes all the names in $\mathrm{supp}(g) \setminus \mathrm{supp}(\dot{\pi}_2(g(c)))$. □

Proposition 3.4.31. *If A is an object of $\mathbf{Ass}_S(P)$ and A has computable support, then Σ_A^* is a fibred left adjoint to W_A.*

Proof. The previous proposition shows that Σ_A^* is a fibred functor. It suffices to show that all the morphisms constructed in the proof of Proposition 3.3.32 are tracked by some realizer. First, the unit of the adjunction is defined by $\eta_f(b) = \langle [b]_f, f_2(b) \rangle$. It is tracked by $\lambda x.\mathit{pair}\ x\ (\mathit{snd}\ (r_f\ x))$, where r_f is a realizer for f, which exists since f is a map in $\mathbf{Ass}_S(P)$. The adjoint transpose $u^\sharp$ of a map u is defined by $u^\sharp([b]_f) = \dot{\pi}_1(u(b))$. It is tracked by $\lambda x.\mathit{fst}\ (r_u\ x)$, where r_u is a realizer for u, which exists since u is a map in $\mathbf{Ass}_S(P)$. □

We note that the assumption in this proposition that A have computable support is indeed necessary.

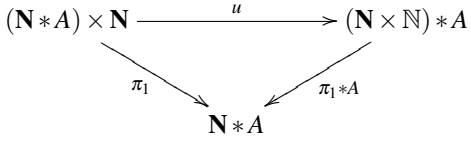
Proposition 3.4.32. *If Σ_A^* is a fibred left adjoint to $W_A \colon \mathit{cod} \to \mathit{Gl}(- * A)$ then A has computable support.*

Proof. Since the proof is a little technical, we first explain the idea intuitively. Let A be an object of $\mathbf{Ass}_S(P)$ and $x \in |A|$. We want to compute the support of x. Since we can find a finite approximation of the support, it suffices to show that, for any given name $n \in |\mathbf{N}|$, it can be decided whether or not $n \in \mathrm{supp}(x)$ holds. Such a decision procedure can be obtained using $\Sigma_A^*\mathbf{N}$. Recall from the Introduction the terms $(M.N)$ and $(\text{let } M \text{ be } x.y \text{ in } N)$ for Σ^* and their informal interpretation. The following term in this notation decides whether or not $n \in \mathrm{supp}(x)$ holds.

$$\text{let } x.n \text{ be } x'.n' \text{ in } (\text{if } n = n' \text{ then yes else no})$$

This term can be understood as follows. First we take the pair $x.n$, which is essentially the pair $\langle x, n \rangle$ but with all the names of x hidden. Then, we exchange the hidden names, but only those, in this pair with fresh names, and match the result against the pair $x'.n'$. Finally, we return 'yes' if $n = n'$ and 'no' otherwise. This function returns 'yes' if and only if n is in the support of x. To see this note that if $n \in \mathrm{supp}(x)$ holds then the name n is hidden (or bound) in $x.n$. Since the pair $x'.n'$ is obtained by freshening all the hidden names, this implies $n \neq n'$. On the other hand, if $n \notin \mathrm{supp}(x)$ then n is not hidden in $x.n$. Since we freshen only hidden names, this implies $n = n'$.

Let A be an object of $\mathbf{Ass}_S(P)$ and assume that Σ_A^* is a fibred left adjoint to W_A. We show that A has computable support. Consider the map u in $\mathbf{Ass}_S(P)/(\mathbf{N} * A)$

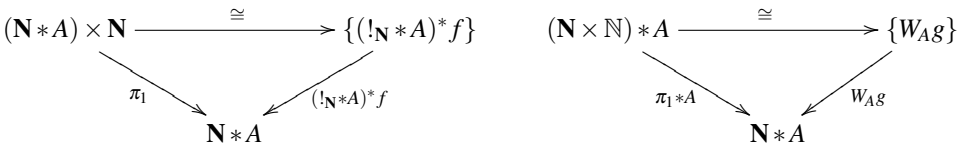


defined by

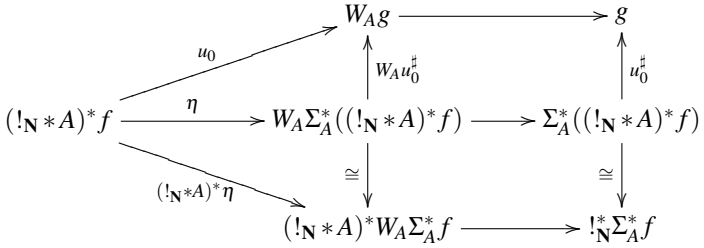
$$u(n,a,m) = \begin{cases} \langle n,0,a\rangle & \text{if } n = m \\ \langle n,1,a\rangle & \text{if } n \neq m. \end{cases}$$

The codomain of u is, by definition of W_A, just $W_A(\pi_1)$. We show that the adjoint transpose of u can be used to compute the support on A.

Let f be the object of $\mathbf{Ass}_S(P)/(1 * A)$ given by the projection $(1 * A) \times \mathbf{N} \to (1 * A)$ and let g be the object of $\mathbf{Ass}_S(P)/\mathbf{N}$ given by the projection $\mathbf{N} \times \mathbb{N} \to \mathbf{N}$. Write $!_{\mathbf{N}} \colon \mathbf{N} \to 1$ for the terminal map. We have the following isomorphisms (the second one is in fact an equality).

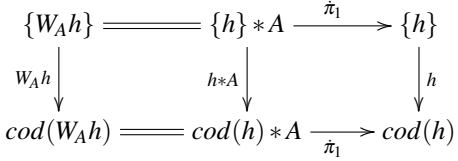


By composing with these isomorphisms, the map u amounts to a map $u_0 \colon (!_{\mathbf{N}} * A)^* f \to W_A g$. Let $u_0^\sharp \colon \Sigma_A^*(!_{\mathbf{N}} * A)^* f \to g$ be the adjoint transpose of u_0. Since $\Sigma_A^* \dashv W_A$ is a fibred adjunction, the following diagram commutes in $\mathbf{Ass}_S(P)^{\to}$.

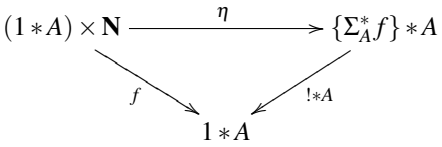


(3.19)

In this diagram, the unlabelled vertical morphisms are the canonical isomorphisms arising since all functors in the diagram are fibred functors. The unlabelled horizontal maps are given by the natural transformation which, for an object h in $\mathbf{Ass}_S(P)^{\to}$, is the map $W_A h \to h$ given by the following projection.

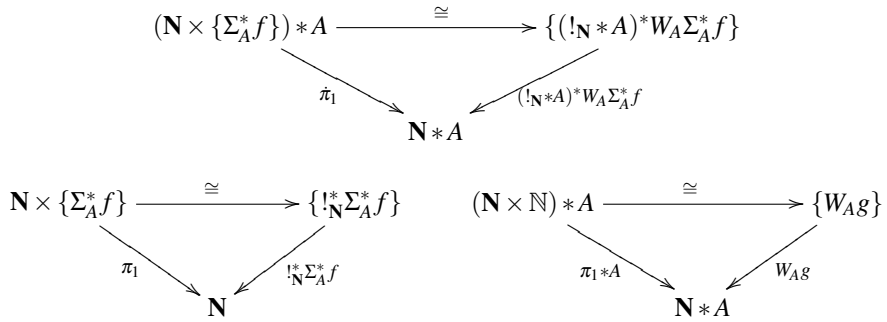


In the above diagram, the unit $\eta \colon f \to W_A \Sigma_A^* f$ is a map of the form

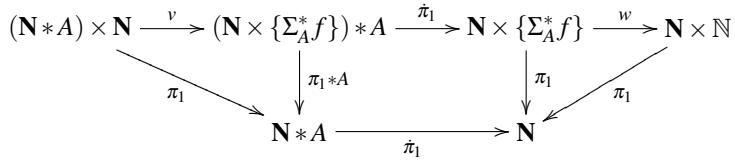


For $a \in A$ and $n \in \mathbf{N}$, we write $a.n$ for $\eta(\langle \diamond, a\rangle, n)$.

For the objects in the diagram (3.19) we have the following isomorphisms.



Using these isomorphisms the path $(!_{\mathbf{N}}*A)^* f \to (!_{\mathbf{N}}*A)^* W_A\Sigma_A^* f \to !_{\mathbf{N}}^*\Sigma_A^* f \to g$ in the diagram (3.19) amounts to a map



with $v(m,a,n) = \langle m,(a.n),a\rangle$ and $w\circ\dot{\pi}_1\circ v = \dot{\pi}_1\circ u$. It follows easily from these two equations that, for all $n,m\in\mathbf{N}$ and all $a\in A$ such that a is fresh for m, it holds that $w(m,(a.n)) = \langle m,u(m,a,n)\rangle$. Notice that w is, up to composition with isomorphism, just $u^\sharp$.

Using the map w, we now show

$$\forall a\in A.\,\forall n\in\mathbf{N}.\quad n\in\mathrm{supp}(a)\iff \pi_1(w(n,a.n)) = 1.$$

Let $a\in A$ and $n\in\mathbf{N}$. If $n\notin\mathrm{supp}(a)$ then $n\,\#\,a$ and therefore

$$\pi_2(w(n,(a.n))) = \pi_2(\dot{\pi}_1(u(n,a,n))) = 0.$$

Suppose $n\in\mathrm{supp}(a)$. Let τ be a permutation that exchanges each name in $\mathrm{supp}(a)$ with a fresh name and leaves all other names fixed. Then, $n\,\#\,\tau\cdot a$ and $\tau\cdot n\neq n$. Since the codomain of v is $(\mathbf{N}\times\{\Sigma_A^* f\})*A$, we have $a\,\#\,(a.n)$, which implies $a.n = \tau\cdot(a.n)$. Using this we have

$$\pi_2(w(n,(a.n))) = \pi_2(w(n,\tau\cdot(a.n))) = \pi_2(w(n,(\tau\cdot a.\tau\cdot n))) = \pi_2(\dot{\pi}_1(u(n,\tau\cdot a,\tau\cdot n))) = 1.$$

It therefore follows that the map $\lambda a\colon A.\,\lambda n\colon\mathbf{N}.\,\pi_2(w(n,(a.n)))$, which is easily seen to be tracked, can be use to decide whether or not a given name $n\in\mathbf{N}$ is in the support of an element $a\in A$. Since $\mathbf{Ass}_S(P)$ has finite support approximations, we can therefore compute the support of an element a of A by first approximating its support and then testing which names in this approximation really belong to the support of a. □

A consequence of this proposition is that not all Σ^*-types need to exist in a codomain fibration. This is in contrast to Σ-types, which in a codomain fibration always exist.

Corollary 3.4.33. *The category* $\mathbf{Ass}_S(P)$ *constructed over the Kleene PCA does not have all* Σ^**-types.*

Proof. First, we show that there exists an object that does not have computable support. We show that $\mathbb{N} \Rightarrow \mathbf{N}$ is such an object. Let K be a code for a Turing machine and let $n_0, n_1 \in \mathbf{N}$ be two names with $n_0 \neq n_1$. Consider the following function.

$$f_K(m) = \begin{cases} n_0 & \text{if } K \text{ stops in no more than } m \text{ steps} \\ n_1 & \text{otherwise} \end{cases}$$

We have $\mathrm{supp}(f_K) = \{n_0, n_1\}$ if K stops and $\mathrm{supp}(f_K) = \{n_1\}$ if K never stops. Hence, the code $\lambda\!\!\lambda x.[\overline{n_0}, \overline{n_1}]$ is a support approximation for f_K. Therefore, given an (effective) encoding K of a Turing machine, we can effectively compute a code for f_K in $\mathbf{Ass}_S(P)$. However, the object $\mathbb{N} \Rightarrow \mathbf{N}$ cannot have computable support, since if we were able to compute the support on $\mathbb{N} \Rightarrow \mathbf{N}$ exactly, then we could solve the halting problem. Proposition 3.4.32 implies that $\mathbf{Ass}_S(P)$ does not have $\Sigma^*_{\mathbb{N}\Rightarrow\mathbf{N}}$-types. □

That $\mathbf{Ass}_S(P)$ has Σ^*_A-types only when A has computable support should be expected. After all, $\Sigma^*_A B$ consists of pairs $a.b$, where $a \in A$ and $b \in B$, in which all the names in a have been hidden. It is not surprising that if we are to hide the names in a, then we must at least be able to compute them. The restriction that Σ^*_A can only be formed for A with computable support also appears in FreshML. There, the abstraction set $[A]B$, corresponding to $\Sigma^*_A B$, can only be formed if A is an equality type, i.e. has decidable equality. We have seen above that all types with decidable equality have computable support.

We have given the definition of Σ^*_A for the non-split codomain fibration, and we have seen in the proof that Σ^*_A is a fibred functor that the isomorphism $u^*\Sigma^*_A \cong \Sigma^*(u * A)^*$ requires us to 'freshen' the elements of Σ^*_A. We should point out that eventually we will be using a split version of Σ^*, and, since for split fibred functors there is an equality $u^*\Sigma^*_A = \Sigma^*(u * A)^*$, this freshening of equivalence classes will be part of the action of Σ^*_A on morphisms. Obtaining a split fibration is the topic of the next section.

3.5 Splitting the Codomain Fibration

Having constructed the relevant categorical structure in the codomain fibrations on $\mathbb{S}$ and $\mathbf{Ass}_S(P)$, we now come to constructing equivalent split fibrations in which the syntax can be interpreted more easily. As we have discussed in Section 2.5.3, for the commonly used splitting using the fibred Yoneda lemma it is not obvious how to give a split version of all the structure, in particular strong monoidal products Π^*. In this section we show that for quasi-toposes we can define a split version of all the structure using the splitting construction of [56, §10.5.9]. In essence, the interpretation of dependent types in this split fibration corresponds to the well-known set-theoretic interpretation.

3.5.1 Split Closed Comprehension Categories for Quasi-toposes

For each quasi-topos $\mathbb{B}$ we build a split closed comprehension category that is equivalent to the codomain fibration $\mathbb{B}^{\to} \to \mathbb{B}$. For toposes this construction appears in [56, §10.5.9]; we observe here that it continues to work for quasi-toposes. Furthermore, we give a construction of split versions of W_A and Π_A^* from a (certain) monoidal closed structure.

Let $\mathbb{B}$ be a quasi-topos. We define the split *family fibration* $\mathscr{F}\colon \mathscr{F}(\mathbb{B}) \to \mathbb{B}$. We assume that we have, for each map $\varphi\colon X \to \Omega$, a choice of strong monomorphism $m_\varphi\colon \{\varphi\} \rightarrowtail X$ classified by φ. Such a choice is available in all the categories considered so far, since in all these categories we have a canonical construction of pullbacks.

The total category $\mathscr{F}(\mathbb{B})$ of $\mathscr{F}$ is defined as follows.

Objects. An object of $\mathscr{F}(\mathbb{B})$ is a triple $(\Gamma, X, \varphi\colon \Gamma \times X \to \Omega)$. Each such object determines a projection map $\pi_\varphi = \pi_1 \circ m_\varphi\colon \{\varphi\} \rightarrowtail \Gamma \times X \to \Gamma$. We will frequently write just φ for the triple (Γ, X, φ) when Γ and X are clear from the context.

Morphisms. A morphism from (Γ, X, φ) to (Δ, Y, ψ) is a map from π_φ to π_ψ in $\mathbb{B}^{\to}$.

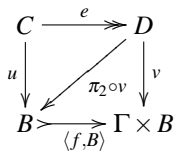
The evident projection functor $\mathscr{F}\colon \mathscr{F}(\mathbb{B}) \to \mathbb{B}$ mapping the object (Γ, X, φ) to Γ is a fibration. The cartesian maps for this fibration are those maps of $\mathscr{F}(\mathbb{B})$ that are given by pullback squares. The fibration becomes a split fibration if, for a map $u\colon \Gamma \to \Delta$ in $\mathbb{B}$ and an object (Δ, X, φ) over Δ, we choose the cartesian lifting to be the unique morphism $(\overline{u}(\varphi), u)$ making the diagram below commute. This means that reindexing is given by pre-composition: the object $u^*\varphi$ is just $\varphi \circ (u \times X)\colon \Gamma \times X \to \Omega$.

$$\begin{array}{ccc}
\{\varphi \circ (u \times X)\} & \xrightarrow{\overline{u}(\varphi)} & \{\varphi\} \\
{\scriptstyle m_{\varphi\circ(u\times X)}}\downarrow & \lrcorner & \downarrow{\scriptstyle m_\varphi} \\
\Gamma \times X & \xrightarrow[u\times X]{} & \Delta \times X \\
{\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle \pi_1} \\
\Gamma & \xrightarrow[u]{} & \Delta
\end{array}$$

Proposition 3.5.1. *The fibration $\mathscr{F}(\mathbb{B}) \to \mathbb{B}$ is equivalent to the codomain fibration $cod\colon \mathbb{B}^{\to} \to \mathbb{B}$.*

Proof. One direction of the equivalence is given by the functor $E\colon \mathscr{F}(\mathbb{B}) \to \mathbb{B}^{\to}$ mapping (Γ, X, φ) to the projection $\pi_\varphi\colon \{\varphi\} \to \Gamma$. On morphisms E, is the identity since the morphisms $\varphi \to \psi$ in $\mathscr{F}(\mathbb{B})$ are, by definition, already morphisms $\pi_\varphi \to \pi_\psi$ in $\mathbb{B}^{\to}$. In the reverse direction, an object $f\colon B \to \Gamma$ is mapped to the triple (Γ, B, φ) where φ is the unique morphism of type $\Gamma \times B \to \Omega$ that classifies $\langle f, B\rangle\colon B \rightarrowtail \Gamma \times B$. For this we have to check that $\langle f, B\rangle$ is a strong monomorphism. This is easily seen, since, for all $u\colon C \to B$, $v\colon D \to \Gamma \times B$ and $e\colon C \twoheadrightarrow D$, the whole diagram below commutes whenever the

outer square of it commutes.



The rest of the proof is as in [56, §10.5.9]. □

The split fibration $\mathscr{F}(\mathbb{B}) \to \mathbb{B}$ generalises the family fibration on sets, as illustrated by the following examples.

Examples 3.5.2.

1. The fibration $\mathscr{F}(\mathbf{Sets}) \to \mathbf{Sets}$ is equivalent to the family fibration $\mathrm{Fam}(\mathbf{Sets}) \to \mathbf{Sets}$ on sets. The total category $\mathrm{Fam}(\mathbf{Sets})$ has as objects set-indexed families of sets $(X_i)_{i\in I}$. The morphisms from $(X_i)_{i\in I}$ to $(Y_j)_{j\in J}$ are pairs $(u,(f_i)_{i\in I})$ where u is a function of type $I \to J$ and, for each $i \in I$, f_i is a function of type $X_i \to Y_{u(i)}$. The fibration $\mathscr{F}$ is given by the projection functor mapping an object $(X_i)_{i\in I}$ to I and a morphism $(u,(f_i)_{i\in I})$ to u.

2. The fibration $\mathscr{F}(\mathbb{S}) \to \mathbb{S}$ may be described as the following family fibration $\mathrm{Fam}(\mathbb{S}) \to \mathbb{S}$. The total category $\mathrm{Fam}(\mathbb{S})$ has as objects pairs $(I,(X_i)_{i\in I})$ where I is an object of $\mathbb{S}$ and $(X_i)_{i\in I}$ is a set-indexed family which comes equipped with a permutation action on the union $X \stackrel{\text{def}}{=} \bigcup_{i\in I} X_i$, such that X has the finite support property and that, for each permutation π, the equality $\pi\cdot(X_i) = X_{\pi\cdot i}$ holds. The morphisms from $(I,(X_i)_{i\in I})$ to $(J,(Y_j)_{j\in J})$ are pairs $(u,(f_i)_{i\in I})$, where u is a morphism of type $I \to J$ in $\mathbb{S}$ and, for each $i \in I$, f_i is a function of type $X_i \to Y_{u(i)}$, satisfying $\pi\cdot f_i(x) = f_{\pi\cdot i}(\pi\cdot x)$. The fibration is again given by the projection functor mapping an object $(I,(X_i)_{i\in I})$ to I and a morphism $(u,(f_i)_{i\in I})$ to u.

3. The fibration $\mathscr{F}(\mathbf{Ass}) \to \mathbf{Ass}$ may be described as the fibration $\mathrm{UFam}(\mathbf{Ass}) \to \mathbf{Ass}$ of uniform families of assemblies. The total category $\mathrm{UFam}(\mathbf{Ass})$ has as objects pairs $(I,(X_i)_{i\in|I|})$ where I is an object in $\mathbf{Ass}$ and $(X_i)_{i\in|I|}$ is a set-indexed family of assemblies. A morphism from $(I,(X_i)_{i\in|I|})$ to $(J,(Y_j)_{j\in|J|})$ is a pair $(u,(f_i)_{i\in|I|})$ where u is a morphism in $\mathbf{Ass}$ and $(f_i)_{i\in|I|}$ is a set-indexed family of functions $f_i\colon X_i \to Y_{u(i)}$ which is uniformly realizable, meaning that there exists a realizer $r \in \mathscr{X}$ such that for each $i \in |I|$ and each $s \in \|i\|_I$, $r\ s$ is defined and realizes f_i.

More information on the fibrations in 1 and 3 can be found in [56]. □

We prefer to work with the general description of the fibration $\mathscr{F}(\mathbb{B}) \to \mathbb{B}$, since we find it simpler than the specific descriptions above. For example, an explicit description of the fibration $\mathscr{F}(\mathbf{Ass}_S(P)) \to \mathbf{Ass}_S(P)$ is a combination of the points 2 and 3 above and comes out rather complicated.

We consider the construction of dependent sums and products from [56, §10.5.9]. The following definitions of $\Sigma_\varphi\psi$ and $\Pi_\varphi\psi$ are exactly the same as [56, §10.5.9], where they are given for toposes.

Here we just observe that these constructions still work for quasi-toposes. Let $\varphi\colon \Gamma \times X \to \Omega$ and $\psi\colon \{\varphi\} \times Y \to \Omega$ be objects over Γ and $\{\varphi\}$ respectively. The dependent sum $\Sigma_\varphi \psi$ over Γ is constructed as the unique characteristic map in the diagram

$$\begin{array}{ccccccc}
\{\psi\} & \xrightarrow{m_\psi} & \{\varphi\} \times Y & \xrightarrow{m_\varphi \times Y} & (\Gamma \times X) \times Y & \xrightarrow{\cong} & \Gamma \times (X \times Y) \\
{\scriptstyle !}\downarrow & & & & & & \downarrow{\scriptstyle \Sigma_\varphi \psi} \\
1 & & \xrightarrow[true]{} & & & & \Omega
\end{array}$$

Notice that the top row of this diagram is a strong monomorphism since strong monomorphisms are preserved by products and closed under composition and isomorphisms are strong. The verification that this indeed defines a dependent sum is routine.

For the product $\Pi_\varphi \psi$ we use weak partial map classifiers in the quasi-topos. Let $\tilde{\varphi}$ and $\tilde{\psi}$ be defined as the unique maps making the following squares pullbacks:

$$\begin{array}{ccc}
\{\varphi\} & \xrightarrow{m_\varphi} & \Gamma \times X \\
\| & & \downarrow{\scriptstyle \tilde{\varphi}} \\
\{\varphi\} & \xrightarrow[\eta_\varphi]{} & \{\varphi\}_\perp
\end{array}$$

$$\begin{array}{ccccc}
\{\psi\} & \xrightarrow{m_\psi} & \{\varphi\} \times Y & \xrightarrow{m_\varphi \times \eta_Y} & (\Gamma \times X) \times Y_\perp \\
\| & & & & \downarrow{\scriptstyle \tilde{\psi}} \\
\{\psi\} & & \xrightarrow[\eta_\psi]{} & & \{\psi\}_\perp
\end{array}$$

Note that all the monomorphisms in these diagrams are strong, since the weak partial map classifiers η are all strong. Define maps

$$\begin{array}{rcll}
\alpha & = & \langle \pi_1 \times X, \mathrm{ev} \circ (\pi_2 \times X) \rangle & : (\Gamma \times (X \Rightarrow Y_\perp)) \times X \to (\Gamma \times X) \times Y_\perp \\
\beta & = & \Lambda(\perp(\pi_2 \circ m_\varphi) \circ \tilde{\varphi}) \circ \pi_1 & : \Gamma \times (X \Rightarrow Y_\perp) \to (X \Rightarrow X_\perp) \\
\gamma & = & \Lambda(\perp(\pi_2 \circ m_\varphi \circ \pi_\psi) \circ \tilde{\psi} \circ \alpha) & : \Gamma \times (X \Rightarrow Y_\perp) \to (X \Rightarrow X_\perp)
\end{array}$$

and define the product $\Pi_\varphi \psi\colon \Gamma \times (X \Rightarrow Y_\perp) \to \Omega$ to be the characteristic map of the equaliser e.

$$\Pi_\varphi \psi \xrightarrow{e} \Gamma \times (X \Rightarrow Y_\perp) \underset{\gamma}{\overset{\beta}{\rightrightarrows}} (X \Rightarrow X_\perp)$$

The proofs that these constructions define split sums and split products are routine. The construction of Π amounts to the proof in [117, §18] that the local cartesian closed structure of a quasi-topos can be constructed from weak partial map classifiers.

We come to the monoidal structure. Let $\mathbb{B}$ have a strict affine symmetric monoidal closed structure $(*, -\!\!*)$ with the property that, for each object A, the functor $- * A\colon \mathbb{B} \to \mathbb{B}$ preserves pullbacks and that, for all objects A and B of $\mathbb{B}$, the canonical inclusion $A * B \rightarrowtail A \times B$ is strong. Notice that the monoidal structures in both $\mathbb{S}$ and $\mathbf{Ass}_S(P)$ have this property.

First we give a split version of W_A. Let $\varphi\colon \Gamma \times B \to \Omega$ be an object over Γ. We define the object $W_A(\varphi)\colon (\Gamma * A) \times B \to \Omega$ to be the unique characteristic map making the square below a pullback.

$$\begin{array}{ccccc}
\{\varphi\} * A & \xrightarrow{m_\varphi * A} & (\Gamma \times B) * A & \xrightarrow{\langle \pi_1 * A, \pi_2 \circ \pi_1 \rangle} & (\Gamma * A) \times B \\
\downarrow & \lrcorner & & & \downarrow{\scriptstyle W_A\varphi} \\
1 & & \xrightarrow[true]{} & & \Omega
\end{array}$$

It is easily seen that the top row in this diagram is a strong monomorphism.

For the definition of the morphism part of W_A, we note that the definition of the object $W_A(\varphi)$ gives a canonical isomorphism $i_\varphi\colon \pi_{W_A(\varphi)} \cong \pi_\varphi * A$ in the slice category $\mathbb{B}/(\Gamma * A)$. Given a morphism $u\colon \pi_\varphi \to \pi_\psi$ in $\mathbb{B}^{\to}$, the morphism action $W_A(u)$ is then defined by:

$$\begin{array}{ccc}
\pi_{W_A(\varphi)} & \xrightarrow[\cong]{i_\varphi} & \pi_\varphi * A \\
{\scriptstyle W_A(u)}\downarrow & & \downarrow{\scriptstyle u * A} \\
\pi_{W_A(\psi)} & \xrightarrow[i_\psi]{\cong} & \pi_\psi * A
\end{array}$$

With this definition, W_A becomes naturally isomorphic to the non-split version of W_A defined in Section 2.5.1. In particular, since $- * A$ preserves pullbacks by assumption, it follows that W_A preserves cartesian maps, i.e. it is a fibred functor from $\mathscr{F}$ to $(- * A)^*\mathscr{F}$.

To show that the thus defined W_A is a split functor, it remains to verify that, for any morphism $u\colon \Delta \to \Gamma$ in $\mathbb{B}$ and any object $\varphi\colon \Gamma \times B \to \Omega$ over Γ, the morphisms $W_A(\overline{u}(\varphi))$ and $\overline{(u * A)}(W_A\varphi)$ are the same. The morphism $W_A(\overline{u}(\varphi))$ is, by definition, given by the top row in the diagram below. To show the required equality, we first show that the outer part of this diagram is a pullback.

$$\begin{array}{ccccccc}
\{W_A(\varphi \circ (u \times B))\} & \xrightarrow[\cong]{i_{\varphi\circ(u\times B)}} & \{\varphi \circ (u \times B)\} * A & \xrightarrow{\overline{u} * A} & \{\varphi\} * A & \xrightarrow[\cong]{i_\varphi^{-1}} & \{W_A(\varphi)\} \\
 & {\scriptstyle m_{W_A(\varphi\circ(u\times B))}}\searrow & \downarrow{\scriptstyle m_{\varphi\circ(u\times B)} * A} & \lrcorner & \downarrow{\scriptstyle m_\varphi * A} & & \swarrow{\scriptstyle m_{W_A(\varphi)}} \\
 & & (\Delta \times B) * A & \xrightarrow{(u\times B) * A} & (\Gamma \times B) * A & & \\
 & & \downarrow & \lrcorner & \downarrow & & \\
 & & (\Delta * A) \times B & \xrightarrow[(u * A) \times B]{} & (\Gamma * A) \times B & &
\end{array} \tag{3.20}$$

That the topmost inner square is a pullback follows immediately since $- * A$ preserves pullbacks. That the lower square is a pullback follows because in the diagram below the outermost square is a pullback, by pullback preservation of $- * A$, and the lowermost square is a pullback, by properties of the cartesian

product, so that the topmost square must also be a pullback.

$$\begin{array}{ccc}
(\Delta \times B) * A & \xrightarrow{(u\times B)*A} & (\Gamma \times B) * A \\
\downarrow & & \downarrow \\
(\Delta * A) \times B & \xrightarrow[(u*A)\times B]{} & (\Gamma * A) \times B \\
{\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle \pi_1} \\
\Delta * A & \xrightarrow[u]{} & \Gamma * A
\end{array}$$

Since both inner squares in diagram (3.20) are pullbacks and since i_φ^{-1} and $i_{\varphi\circ(u\times B)}$ are isomorphisms, it follows that the outer part of diagram (3.20) is a pullback. Hence, because $W_A(\varphi)\colon (\Gamma * A)\times B \to \Omega$ and $W_A(\varphi\circ(u\times B))\colon (\Delta * A)\times B \to \Omega$ are the characteristic maps of $m_{(W_A(\varphi))}$ and $m_{(W_A(\varphi\circ(u\times B)))}$ respectively, we have $W_A(\varphi\circ(u\times B)) = W_A(\varphi)\circ((u*A)\times B)$. Since reindexing is defined by precomposition, this implies the equality of the objects $W_A(u^*\varphi) = (u*A)^*W(\varphi)$. Furthermore, since $m_{W_A(\varphi)}$ is monomorphic, any map from $\{W_A(\varphi\circ(u\times B))\}$ to $\{W_A(\varphi)\}$ making the outermost square of (3.20) commute must be equal to the top row of (3.20). By definition, $\overline{(u*A)}(W(\varphi))$ is such a map, so that, since the top row of the diagram is just $W_A(\overline{u}(\varphi))$, we have $W_A(\overline{u}(\varphi)) = \overline{(u*A)}(W(\varphi))$. This shows that W_A preserves the splitting.

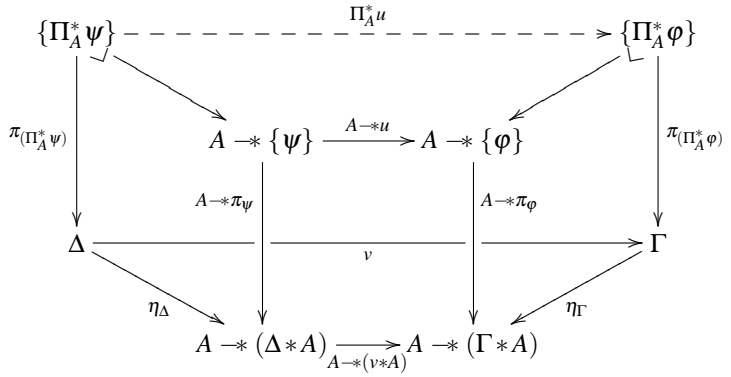
Next we come to the construction of a split version of Π^*. Let $\varphi\colon (\Gamma * A)\times B \to \Omega$ be an object over $\Gamma * A$ in $\mathscr{F}$. First note that $A \multimap -$, being a right adjoint, preserves equalisers and therefore by Proposition 3.1.11 also strong monomorphisms. Note also that strong monomorphisms are closed under pullback. These two facts imply that all the monomorphisms in the diagram below are strong. Define the object $\Pi_A^*\varphi\colon \Gamma\times(A\multimap B)\to\Omega$ over Γ to be the classifying map of the strong monomorphism m given by the following pullback.

$$\begin{array}{ccc}
C & \longrightarrow & A \multimap \{\varphi\} \\
{\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle A\multimap m_\varphi} \\
\Gamma\times(A\multimap B) & \xrightarrow[\alpha]{} & A\multimap((\Gamma * A)\times B) \\
{\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle A\multimap\pi_1} \\
\Gamma & \xrightarrow[\eta]{} & A\multimap(\Gamma * A)
\end{array} \tag{3.21}$$

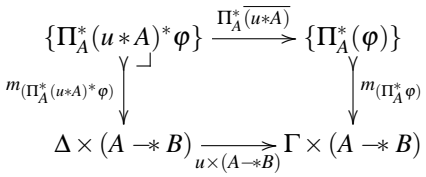
In this diagram, α is the exponential transpose of $\langle \pi_1 * A, \varepsilon\circ(\pi_2 * A)\rangle\colon (\Gamma\times(A\multimap B))*A \to (\Gamma * A)\times B$. The lower square is a pullback: given two maps $f\colon C\to\Gamma$ and $g\colon C\to(A\multimap((\Gamma * A)\times B))$ that satisfy $\eta\circ f = (A\multimap\pi_1)\circ g$, we take $h = \langle f, (A\multimap\pi_2)\circ g\rangle\colon C\to\Gamma\times(A\multimap B)$ as the pullback mediator. The verifications are routine.

Hence, diagram (3.21) is a pullback, which immediately shows that Π^* as defined above is, up to the equivalence of fibrations *Fib* and *cod*, naturally isomorphic to the non-split version of Π^* defined by the same pullback (2.1). We use this fact to define the morphism part of Π^* and to obtain that it is indeed a fibred functor from $(-*A)^*\mathscr{F}$ to $\mathscr{F}$, just as we have done for W_A above. Explicitly, a

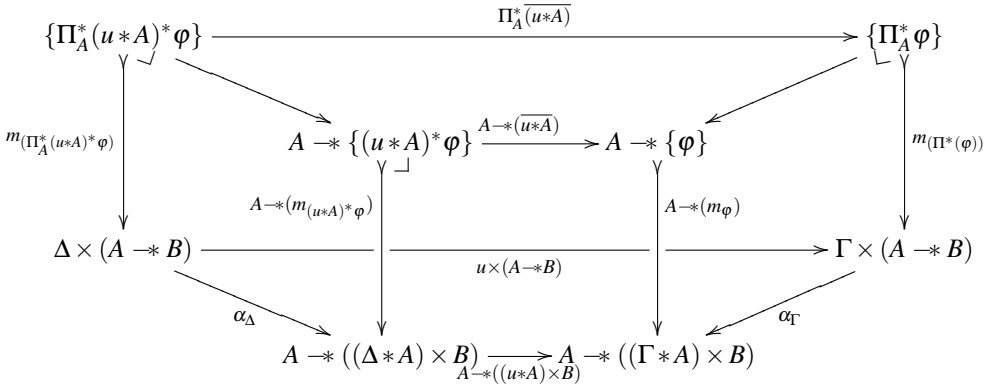
morphism $u\colon \psi \to \varphi$ over $v * A\colon \Delta * A \to \Gamma * A$ is mapped by Π_A^* to the unique map $\Pi_A^* u$ over v making the following diagram commute.



To see that Π_A^* is a split fibred functor, it suffices to show that, for each morphism $u\colon \Delta \to \Gamma$ and each object $\varphi\colon (\Gamma * A) \times B \to \Omega$ over $\Gamma * A$, the square



is a pullback. That this is a pullback can be seen by considering the diagram



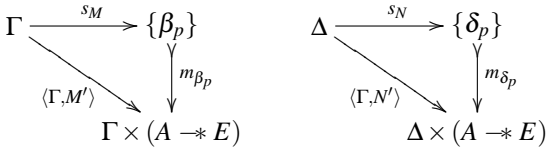
in which α_Δ and α_Γ are defined in the same way as α above. The diagram commutes by definition. The three squares in the front are all pullbacks, by definition and since $A \multimap -$, being a right adjoint, preserves pullbacks. By the pullback lemma, this implies that the square in the background is a pullback.

Proposition 3.5.3. *The functor Π^* as defined above gives strong split simple monoidal products for the fibration $\mathscr{F}(\mathbb{B}) \to \mathbb{B}$.*

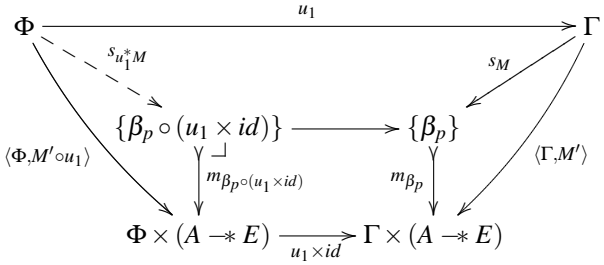
Proof. We have already shown that Π^* defines split simple monoidal products. It just remains to show them strong. Assume objects B in $\mathbb{E}_{\Gamma * A}$ and D in $\mathbb{E}_{\Delta * A}$ and maps $M\colon 1_\Gamma \to \Pi_A^* B$ in $\mathbb{E}_\Gamma$, $N\colon 1_\Delta \to \Pi_A^* D$

in $\mathbb{E}_\Delta$, $u\colon \Phi \to \Gamma * A$ in $\mathbb{B}$ and $v\colon \Phi \to \Delta * A$ in $\mathbb{B}$ that satisfy $u^*B = v^*D$, $(\dot\pi_1 \circ u)^*M = (\dot\pi_1 \circ v)^*N$ and $\dot\pi_2 \circ u = \dot\pi_2 \circ v$. We have to show $u^*(\mathrm{app}^*M) = v^*(\mathrm{app}^*N)$, where app^* is defined as in Section 2.5.2. We use the abbreviations $u_i \stackrel{\text{def}}{=} \dot\pi_i \circ u$ and $v_i \stackrel{\text{def}}{=} \dot\pi_i \circ v$ for $i = 1,2$.

The objects B and D are given by maps $\beta\colon (\Gamma * A) \times B' \to \Omega$ and $\delta\colon (\Delta * A) \times D' \to \Omega$. The objects u^*B and v^*D are given by $\beta \circ (u \times B')$ and $\delta \circ (v \times D')$ respectively. Since $u^*B = v^*D$ holds by assumption, we have $\beta \circ (u \times B') = \delta \circ (v \times D')$, so that $B' = D'$ holds. Let $E \stackrel{\text{def}}{=} B' = D'$. The objects $\Pi_A^* B$ and $\Pi_A^* D$ are defined to be morphisms $\beta_p\colon \Gamma \times (A \multimap E) \to \Omega$ and $\delta_p\colon \Delta \times (A \multimap E) \to \Omega$. As in any comprehension category, the maps M and N correspond uniquely to sections s_M and s_N of π_{β_p} and π_{δ_p} respectively. These sections are uniquely determined by maps M' and N' as in the triangles below.

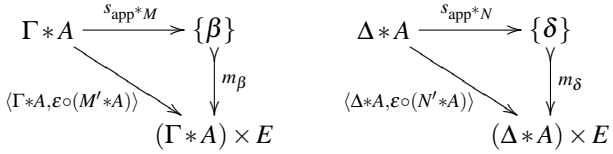


By definition of reindexing, the map u_1^*M corresponds uniquely to a section $s_{u_1^*M}$ of $\pi_{\delta_p \circ (u_1 \times id)}$ for which the following diagram commutes.



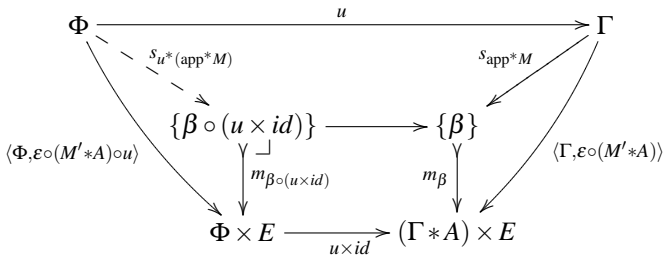
For v_1^*N we have an analogous diagram. From the assumption $u_1^*M = v_1^*N$, it follows that $s_{u_1^*M} = s_{v_1^*N}$ holds. This implies $M' \circ u_1 = N' \circ v_1$.

By definition, the morphisms $\mathrm{app}^*M\colon 1_{\Gamma * A} \to B$ in $\mathbb{E}_{\Gamma * A}$ and $\mathrm{app}^*N\colon 1_{\Gamma * A} \to D$ in $\mathbb{E}_{\Delta * A}$ correspond uniquely to sections s_{app^*M} and s_{app^*N} of π_β and π_δ respectively making the following triangles commute.

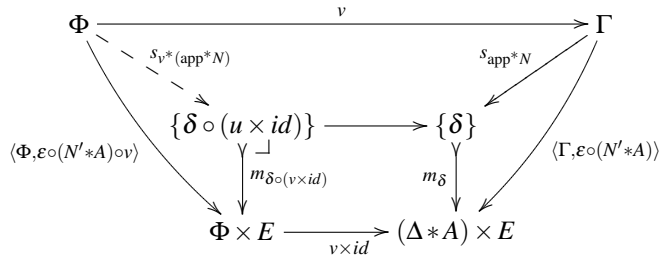


In these diagrams, we write $\varepsilon\colon (A \multimap E) * A \to E$ for the evaluation map. By definition of reindexing,

$u^*(\mathrm{app}^* M)$ corresponds uniquely to a section $s_{u^*(\mathrm{app}^* M)}$ making the following diagram commute.



For $v^*(\mathrm{app}^* N)$ we have an analogous diagram.



To show the required $u^*(\mathrm{app}^* M) = v^*(\mathrm{app}^* N)$, it suffices to $s_{u^*(\mathrm{app}^* M)} = s_{v^*(\mathrm{app}^* N)}$. Since $\beta \circ (u \times B') = \delta \circ (v \times D')$ and since $m_{\beta \circ (u \times B')}$ is monomorphic, it suffices to show $\varepsilon \circ (M' * A) \circ u = \varepsilon \circ (N' * A) \circ v$. Using the equation $M' \circ u_1 = N' \circ v_1$ established above, we get $\dot{\pi}_1 \circ (M' * A) \circ u = M' \circ \dot{\pi}_1 \circ u = N' \circ \dot{\pi}_1 \circ v = \dot{\pi}_1 \circ (N' * A) \circ v$. Furthermore, using the assumption $\dot{\pi}_2 \circ u = \dot{\pi}_2 \circ v$ we get $\dot{\pi}_2 \circ (M' * A) \circ u = \dot{\pi}_2 \circ u = \dot{\pi}_2 \circ v = \dot{\pi}_2 \circ (N' * A) \circ v$. We can therefore make use of the fact that $*$ is a *strict* affine monoidal structure to obtain $(M' * A) \circ u = (N' * A) \circ v$. This implies the required $\varepsilon \circ (M' * A) \circ u = \varepsilon \circ (N' * A) \circ v$, thus completing the proof. □

3.5.2 A Split Fibration for the Schanuel Topos

The above construction of the family fibration $\mathscr{F}(\mathbb{S}) \to \mathbb{S}$ gives us a split closed comprehension category equivalent to the codomain fibration on $\mathbb{S}$ together with split versions of W_A and Π_A^*. It remains to show that Σ_A^* can be turned into a split functor on this split fibration.

First, we spell out the construction of split W_A and Π_A^* in terms of the internal logic of $\mathbb{S}$. For an object $\varphi \colon \Gamma \times B \to \Omega$ over Γ, the object $W_A(\varphi) \colon (\Gamma * A) \times B \to \Omega$ over $\Gamma * A$ is given by the predicate

$$W_A(\varphi)(\gamma, a, b) \iff^{\mathrm{def}} \varphi(\gamma, b) \wedge a \# b.$$

For an object $\psi \colon (\Gamma * A) \times B \to \Omega$ over $\Gamma * A$, the object $\Pi_A^*(\psi) \colon \Gamma \times (A \multimap B) \to \Omega$ over Γ is given by the predicate

$$\Pi_A^*(\psi)(\gamma, f) \iff^{\mathrm{def}} \forall a \in A.\, a \# \langle \gamma, f \rangle \implies \psi(\gamma, a, f(a)) \,.$$

Next we come to defining a split version of Σ^*. To distinguish the split version of Σ^* on the split fibration $\mathscr{F}(\mathbb{S}) \to \mathbb{S}$ from Σ^* on the codomain fibration, we write Σ_s^* for the split version on $\mathscr{F}(\mathbb{S}) \to \mathbb{S}$

and write Σ^* for the non-split version on the codomain fibration. To define Σ^*_s we first fix some notation for the special case of $\Sigma^*_A B$ where B corresponds to a closed type. Let A and B be objects of $\mathbb{S}$. Consider the map $f \stackrel{\text{def}}{=} (\text{unit} \circ \pi_1) \colon A \times B \to 1 * A$ and the object $\Sigma^*_A f$ of $\mathbb{S}/1$. We write $[A]B$ for the domain of $\Sigma^*_A f$, which is defined to be

$$[A]B = \{[a,b]_f \mid a \in A \ \wedge \ b \in B\}.$$

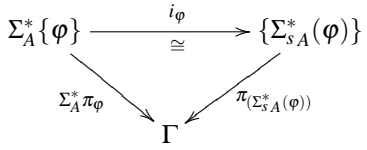
Here, $[a,b]_f$ is the equivalence class of $\langle a,b\rangle$ under the relation $\sim_f$ defined in (3.11). Since f is determined by the objects A and B alone, we also write $[a,b]_{[A]B}$ for $[a,b]_f$.

For an object $\varphi \colon (\Gamma * A) \times B \to \Omega$ over $\Gamma * A$, the object $\Sigma^*_{sA}(\varphi) \colon \Gamma \times [A]B \to \Omega$ over Γ is given by the predicate

$$\Sigma^*_{sA}(\varphi)(\gamma, c) \iff \forall a \in A.\, \forall b \in B.\, (a \# \langle \gamma, c\rangle \ \wedge \ \langle a,b\rangle \in c) \implies \varphi(\gamma,a,b)\,.$$

For the definition of the action on morphisms, we note:

Proposition 3.5.4. *For each object $\varphi \colon (\Gamma * A) \times B \to \Omega$ over $\Gamma * A$, there is an isomorphism i_φ making the following diagram commute.*



Proof. Spelt out, the set $\Sigma^*_A\{\varphi\}$ is given by

$$\Sigma^*_A\{\varphi\} = \{[\gamma,a,b]_{\pi_\varphi} \mid \langle\langle \gamma, a\rangle, b\rangle \in (\Gamma * A) \times B \ \wedge \ \varphi(\gamma,a,b)\}$$

and the map $\Sigma^*_A \pi_\varphi$ maps $[\gamma,a,b]_{\pi_\varphi}$ to γ.

The map i_φ is defined as $i_\varphi([\gamma,a,b]_{\pi_\varphi}) = \langle \gamma, [a,b]_{[A]B}\rangle$. Its inverse i_φ^{-1} maps $\langle \gamma, c\rangle$ to $[\gamma,a,b]_{\pi_\varphi}$, where a and b are such that $c = [a,b]_{[A]B}$ and a is fresh for γ. To show that this definition of i_φ^{-1} defines a function, first note that for the equivalence class c we can always find a representation $[a,b]_{[A]B}$ such that a is fresh for γ. By definition of Σ^*_{sA}, we then have $\varphi(\gamma,a,b)$. For well-definedness of i_φ^{-1} we first show that the choice of representative does not matter. Suppose we have $\langle a,b\rangle \sim_{[A]B} \langle a',b'\rangle$. By (3.12), we can assume $\pi \cdot \langle a,b\rangle = \pi' \cdot \langle a',b'\rangle$ for permutations $\pi \in (\text{supp}(a) \leftrightarrow X)$ and $\pi' \in (\text{supp}(a') \leftrightarrow X)$, where X is a finite set of fresh names. Since X contains fresh names and because we have $\gamma \# a$ and $\gamma \# a'$ by construction, the permutations π and π' act as the identity on all names in the support of γ. Hence, we have $\pi \cdot \langle\langle \gamma, a\rangle, b\rangle = \pi' \cdot \langle\langle \gamma, a'\rangle, b'\rangle$, which gives $[\gamma,a,b]_{\pi_\varphi} = [\gamma,a',b']_{\pi_\varphi}$. This shows that the choice of representative is not relevant. Since i_φ^{-1} is moreover easily seen to be equivariant, i_φ^{-1} therefore defines a morphism in $\mathbb{S}$. Finally, we check that i_φ and i_φ^{-1} are indeed inverses. First we have $i_\varphi^{-1}(i_\varphi([\gamma,a,b]_{\pi_\varphi})) = i_\varphi^{-1}(\gamma,[a,b]_{[A]B}) = [\gamma,a,b]_{\pi_\varphi}$. Second, $i_\varphi(i_\varphi^{-1}(\gamma,c)) = i_\varphi([\gamma,a,b]_{\pi_\varphi})$, where a and b are chosen such that $c = [a,b]_{[A]B}$ and $\gamma \# a$ hold, and further $i_\varphi([\gamma,a,b]_{\pi_\varphi}) = \langle \gamma, [a,b]_{[A]B}\rangle = \langle \gamma, c\rangle$. Hence, i_φ defines an isomorphism. □

Like for the split versions of W_A and Π^*_A, we use this proposition to define the morphism part of Σ^*_{sA}. A morphism u over v in the total category $(-*A)^*\mathscr{F}(\mathbb{S})$, as given by a diagram

$$\begin{array}{ccc}
\{\varphi\} & \xrightarrow{\quad u \quad} & \{\psi\} \\
\downarrow{\scriptstyle m_\varphi} & & \downarrow{\scriptstyle m_\psi} \\
(\Gamma * A) \times B & & (\Delta * A) \times C \\
\downarrow{\scriptstyle \pi_1} & & \downarrow{\scriptstyle \pi_1} \\
\Gamma * A & \xrightarrow[v*A]{} & \Delta * A,
\end{array}$$

is mapped to the unique morphism $\Sigma^*_{sA}u$ making the following diagram in $\mathbb{S}^{\to}$ commute.

$$\begin{array}{ccc}
\Sigma^*_A \pi_\varphi & \xrightarrow[\cong]{i_\varphi} & \pi_{(\Sigma^*_{sA}\varphi)} \\
\downarrow{\scriptstyle \Sigma^*_A u} & & \downarrow{\scriptstyle \Sigma^*_{sA} u} \\
\Sigma^*_A \pi_\psi & \xrightarrow[i_\psi]{\cong} & \pi_{(\Sigma^*_{sA}\psi)}
\end{array}$$

With this definition, Σ^*_{sA} preserves pullbacks because Σ^* does. Hence, Σ^*_{sA} is a fibred functor. If we consider $\{\varphi\}$ and $\{\psi\}$ as subsets of $(\Gamma * A) \times B$ and $(\Delta * A) \times C$ respectively, then the morphism $\Sigma^*_{sA}u\colon \{\Sigma^*_{sA}(\varphi)\} \to \{\Sigma^*_{sA}(\psi)\}$ can be described directly as mapping $\langle \gamma, c\rangle$ to $\langle v(\gamma), [a, \pi_2(u(\gamma,a,b))]\rangle$, where $a \in A$ and $b \in B$ are such that $c = [a,b]_{[A]B}$ and a is fresh for γ.

It remains to check that Σ^*_{sA} is a split fibred functor. Let $v\colon \Gamma \to \Delta$ be a morphism in $\mathbb{S}$ and let $\psi : (\Delta * A) \times C \to \Omega$ be an object over $\Delta * A$. We have to show that the morphisms $\Sigma^*_{sA}(\overline{(v*A)}(\psi))$ and $\overline{v}(\Sigma^*_{sA}\psi)$ are equal. That the objects $\Sigma^*_{sA}((v*A)^*\psi)\colon \Gamma \times [A]B \to \Omega$ and $v^*(\Sigma^*_{sA}\psi)\colon \Gamma \times [A]B \to \Omega$ are equal, follows immediately since both correspond to the predicate χ defined by

$$\chi(\gamma, c) \iff (\forall a \in A.\, \forall b \in B.\, (a \# \langle \gamma, c\rangle \wedge \langle a, b\rangle \in c) \implies \varphi(v(\gamma), a, b))\,.$$

To show the morphisms $\overline{v}(\Sigma^*_{sA}\psi)$ and $\Sigma^*_{sA}(\overline{(v*A)}(\psi))$ equal, we can always assume that all monomorphisms are subset inclusions. The morphism $\overline{v}(\Sigma^*_{sA}\psi)$ maps $\langle \gamma, c\rangle \in \Gamma \times [A]B$ to $\langle v(\gamma), c\rangle$. Since the morphism $\overline{(v*a)}(\psi)$ maps $\langle \gamma, a, b\rangle$ to $\langle v(\gamma), a, b\rangle$, the above explicit description of the morphism action of Σ^*_s implies that $\Sigma^*_{sA}(\overline{(v*A)}(\psi))$ maps $\langle \gamma, c\rangle$ to $\langle v(\gamma), c\rangle$. Hence, Σ^*_{sA} is a split fibred functor.

3.5.3 A Split Fibration for $\mathbf{Ass}_S(P)$

Similar to the argument for $\mathbb{S}$, we can construct a split fibration that is equivalent to the codomain fibration on $\mathbf{Ass}_S(P)$. The construction of the family fibration $\mathscr{F}(\mathbf{Ass}_S(P)) \to \mathscr{F}$ provides us with a split fibration having split W_A and strong split Π^*_A.

For the definition of Σ^*_s, recall that $\mathbf{Ass}_S(P)$ has only those Σ^*_A-types where A has computable support. For such an object A with computable support, the definition of Σ^*_{sA} is just as for the Schanuel topos above. We need to verify that this defines a split version of Σ^* in $\mathbf{Ass}_S(P)$. In addition to the construction

of Σ_s^* for the Schanuel topos, we now have to give realizers for all the maps used in the construction. The only point that requires attention in the definition of these realizers is where we make choices of the representatives of elements of $[A]B$. In the above definition of Σ_s^* for $\mathbb{S}$, we frequently need to find for a given element $c \in [A]B$ a pair $\langle a, b\rangle \in c$ such that a is fresh for some given set of names. Making use of support approximations and the fact that A has computable support, such choices are realized in the same way as in Lemma 3.3.31.

3.6 Related and Further Work

There are a number of directions for further work. In one direction, we can ask for generalisations of the constructions in this chapter. We are, for example, confident that the construction of the Schanuel topos and Π^* and Σ^* goes through if we replace the countably infinite set of names and the finite support property by an uncountably infinite set of names and a countable support property, as used in [36, 86]. More generally, we believe that our construction continues to work for Cheney's generalisation of the Schanuel topos, in which the support is defined not using finite sets of names but by abstract support ideals [20]. Somewhat more ambitiously, one can ask if the construction of the Schanuel topos in Section 3.3 still works if we replace **Sets** by, say, a boolean topos $\mathbb{B}$ with a natural number object. It may also be possible that by means of such a generalisation the constructions of $\mathbb{S}$ and $\mathbf{Ass}_S(P)$ can be unified to become instances of a more general construction. For example, we have arrived at the definition of support approximation (Definition 3.4.18) directly by pragmatic considerations. However, the definition looks like it may be equivalent to stating 'for each element there exists a finite set supporting it' in some internal logic. Some evidence that a construction analogous to that of the Schanuel topos can be generalised from **Sets** to a boolean topos $\mathbb{B}$ is Urban & Tasson's work on formalising nominal reasoning in Isabelle/HOL [113]. Urban & Tasson construct an axiomatic type class of objects with permutation action and the finite support property inside the higher-order logic of Isabelle/HOL, and show that the so constructed objects behave as expected. This construction takes place in the boolean topos underlying Isabelle/HOL and appears very similar to the development in Section 3.3.

Rather than looking for other models, there is a lot of structure left to explore in the categories presented in this chapter. For example, realizability categories are prime examples of models for polymorphism, and we believe that $\mathbf{Ass}_S(P)$ models impredicative polymorphism.

We end by pointing out that in the splitting construction we have not used all the structure of a quasi-topos. The construction is likely to go through for partial cartesian closed categories with equality [96], which may be described as cartesian closed categories with representable regular partial monomorphisms in which regular monos are closed under composition.

Chapter 4

A Bunched Dependent Type Theory

In the previous chapters we were concerned with the categorical structure that forms the basis of our type theory with names and binding. This structure is built around a fibration modelling a dependent type theory with at least dependent sums Σ and products Π. The main additional feature is a monoidal structure $*$ on the base of the fibration, using which we have defined simple monoidal sums Σ^* and simple monoidal products Π^*.

In this chapter, we begin to study a syntax for this categorical structure. We begin with a dependent type theory containing additive Σ and Π-types as well as simple monoidal products Π^*. We choose to add only Π^* at first because the resulting type theory remains relatively close to standard dependent type theory. For instance, the type theory with Π^* is formulated without the need for commuting conversions. From the next chapter onwards, when we add types for the monoidal structure $*$, we have to deal with commuting conversions. Moreover, introducing only Π^* at first makes it easier to compare the type theory to other substructural type theories, such as those in [19, 92], which also extend additive type theory just with multiplicative function spaces.

The purpose of this chapter is to define the type theory $\mathbf{BT}(1, \Sigma, \Pi, \Pi^*)$ and to show some of its basic properties. All the type systems studied in this thesis are extensions of $\mathbf{BT}(1, \Sigma, \Pi, \Pi^*)$.

The main difficulty in defining $\mathbf{BT}(1, \Sigma, \Pi, \Pi^*)$ is the integration of the two function spaces Π and Π^* in a single type theory. For simple types, the problem of integrating two function spaces in a single type theory has been considered by O'Hearn and Pym [80, 91]. O'Hearn and Pym address the question of designing a typed λ-calculus corresponding to a category that is both cartesian closed and monoidal closed. This question is closely related to giving a type theory with both Π and Π^*, since the special case of Π and Π^* in a codomain fibration amounts to a category that is both *locally* cartesian closed and monoidal closed.

O'Hearn and Pym propose to use *bunched contexts*, or *bunches*, to integrate the two closed structures in a simple type theory. The basic idea is to have more than one way of concatenating contexts. In normal simple type theory we can take contexts Γ and Δ and combine them to form a new context Γ, Δ.

This corresponds to the cartesian product of contexts. In bunched contexts the comma is not the only way of combining contexts. For example, there may be a context combination $\Gamma * \Delta$ corresponding to a monoidal structure[1]. These different ways of combining contexts may be combined in any way, so that one can form, for example, the bunch $(((x\colon A, y\colon B) * (z\colon B)), u\colon D) * (v\colon E, w\colon F)$.

To see how bunches can be used to integrate a cartesian closed with a monoidal closed structure, consider the adjoint one-to-one correspondences for the closed structures.

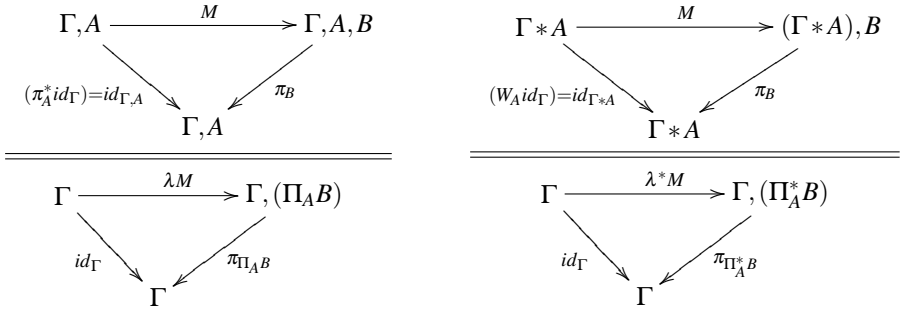
$$\frac{\Gamma \times A \longrightarrow B}{\Gamma \longrightarrow (A \Rightarrow B)} \qquad \frac{\Gamma * A \longrightarrow B}{\Gamma \longrightarrow (A \mathrel{-\!\!*} B)}$$

These correspondences give rise to the following introduction rules.

$$\frac{\Gamma, x\colon A \vdash M : B}{\Gamma \vdash \lambda x\colon A.M : A \Rightarrow B} \qquad \frac{\Gamma * x\colon A \vdash M : B}{\Gamma \vdash \lambda^* x\colon A.M : A \mathrel{-\!\!*} B}$$

The premises of these rules differ only in their use of different context combinations. Bunches are the canonical way of making available in the type theory this difference between these two rules.

Pym and O'Hearn's analysis of bunches as a means of integrating $\Rightarrow$ and $\mathrel{-\!\!*}$ in a simple type theory leads us to use bunches to integrate Π and Π^* in a dependent type theory. A motivating special case of our semantics is that of a category $\mathbb{B}$ that is both locally cartesian closed and monoidal closed. Recall from Proposition 2.5.6 that in this case Π^* is equivalent to monoidal closure. For Π and Π^* we have the following adjoint correspondences.



These correspondences are special cases of the adjoint correspondences for Π and Π^* where we consider only maps with domain id_Γ. The general correspondences can be recovered from these using Lemma 10.4.9 of [56]. The correspondences lead us to the following introduction rules.

$$\frac{\Gamma, x\colon A \vdash M : B}{\Gamma \vdash \lambda x\colon A.M : \Pi x\colon A.B} \qquad \frac{\Gamma * x\colon A \vdash M : B}{\Gamma \vdash \lambda^* x\colon A.M : \Pi^* x\colon A.B}$$

Let us examine the structure of the bunches needed for these two rules. First notice that the object Γ,A is formed using comprehension (the domain functor in this case) on the object π_A in $\mathbb{B}/\Gamma$. Hence, in the context $\Gamma, x\colon A$, the type A should be allowed to depend on the variables in Γ. The comma in the context $\Gamma, x\colon A$ should therefore have the same meaning as in normal dependent type theory. The

[1] Our notation differs from that in [80, 91], where the connectives that we denote by , and $*$ are written as ; and , respectively. We have chosen our notation so that it fits with the standard use of , in dependent type theory as well as with the categorical notation.

object $\Gamma * A$, on the other hand, is formed by multiplying the objects Γ and A of $\mathbb{B}$ using the monoidal structure $*$. Hence, the semantics accounts only for contexts $\Gamma * x\colon A$ in which A does not depend on the variables in Γ. The bunches needed for the above two introduction rules for Π and Π^* must therefore at least have the normal dependent context extension of the form $\Gamma, x\colon A$ as well as a context extension of the form $\Gamma * x\colon A$, in which A is a closed type. Furthermore, we also want to be able to substitute terms for all variables in a context. In particular, we would like to be able to substitute a term M for the variable x in a judgement of the form $\Gamma * x\colon A \vdash \mathscr{J}$. For example, if we have a term $\Gamma \vdash M : \Pi^* x\colon A.B$ then, by adjointness, we get a term $\Gamma * x\colon A \vdash \mathrm{app}^*_{(x:A)B}(M,x) : B$ corresponding to application of the function M to the variable x. If we want to apply M to arguments other than variables then we must be able to substitute a term N for x. A term $\Delta \vdash N : A$ of a closed type A corresponds to a map $\langle N/x\rangle\colon \Delta \to A$ in $\mathbb{B}$. Substituting N for the variable x in a judgement $\Gamma * x\colon A \vdash \mathscr{J}$ then corresponds to substituting along the map $\Gamma * \langle N/x\rangle\colon \Gamma * \Delta \to \Gamma * A$. To account for such substitutions, we need a syntactic representation for the domain of this map. We are therefore lead to considering contexts of the form $\Gamma * \Delta$, where both Γ and Δ are contexts. Note that if we require both Γ and Δ to be contexts then there can be no dependency across the $*$ in $\Gamma * \Delta$.

This train of thought takes us to the simplest form of bunches that can be used to work with Π and Π^* and that allows substitution for all variables. These bunches are built from the empty bunch using the usual dependent context extension $\Gamma, x\colon A$ as well as a non-dependent context multiplication $\Gamma * \Delta$. We discuss other possible choices for the structure of bunches at the end of this chapter.

4.1 The System $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$

In this section we present the bunched type theory $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$. It is a first-order dependent type theory with unit types 1, dependent sums Σ, dependent products Π and simple monoidal products Π^*.

There are many ways of presenting dependent type theories. Definitional equality can be presented as judgemental equality or by means of an external conversion relation. Terms can be monomorphic, i.e. terms have enough type annotations so that types can be inferred from terms, or polymorphic, i.e. the same term may be given different types. Different choices with respect to the presentation lead to different systems whose interrelations are non-trivial [94, 105, 40]. The choices made for the presentation of a dependent type theory have an impact on its suitability for certain purposes. For example, polymorphic presentations with an external conversion relation are arguably well-suited for implementations but less well-suited for studying the relation to categorical formulations, while it is the other way around for monomorphic presentations with judgemental equality. Since in this thesis we are mainly interested in the relation of the type theory to the categorical formulation, we present $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$ as a monomorphic dependent type theory with judgemental equality. Obtaining other presentations of this type theory is an important direction for further work.

4.1.1 Syntax

The syntax of the types and terms is given by the grammar below. We use $x, y, z, \ldots$ to range over a countably infinite set of variables.

$$
\begin{array}{rl}
\text{Contexts} & \Gamma ::= \diamond \mid \Gamma * \Gamma \mid \Gamma, x\colon A \\
\text{Types} & A ::= T(M_1, \ldots, M_n) \mid 1 \mid \Sigma x\colon A.A \mid \Pi x\colon A.A \mid \Pi^* x\colon A.A \\
\text{Terms} & M ::= x \mid f(M_1, \ldots, M_n) \mid \mathrm{unit} \\
& \quad \mid \mathrm{pair}_{(x:A)B}(M,M) \mid \mathrm{fst}_{(x:A)B}(M) \mid \mathrm{snd}_{(x:A)B}(M) \\
& \quad \mid \lambda x\colon A.M \mid \mathrm{app}_{(x:A)B}(M,M) \\
& \quad \mid \lambda^* x\colon A.M \mid \mathrm{app}^*_{(x:A)B}(M,M)
\end{array}
$$

We use $\Gamma, \Delta, \Phi, \ldots$ to range over contexts, $A, B, C, \ldots$ to range over types and $M, N, R, \ldots$ to range over terms. We make the assumption that in a context Γ no variable is declared more than once. We write $v(\Gamma)$ for the ordered list of variables declared in Γ. We do not write the symbol for the empty context when the empty context is extended with the additive context extension, i.e. we write $(x\colon A)$ for $(\diamond, x\colon A)$.

For convenience, we introduce a syntactic category of contexts with exactly one hole.

$$\Gamma_\circ ::= \bigcirc \mid \Gamma_\circ * \Gamma \mid \Gamma * \Gamma_\circ \mid \Gamma_\circ, x\colon A$$

Given a context-with-hole $\Gamma_\circ$ and a context Δ, we write $\Gamma(\Delta)$ for the context obtained by replacing the hole in $\Gamma_\circ$ with Δ. The context $\Gamma(\Delta)$ is again subject to the assumption that no variable is declared more than once.

The set of free variables of types and terms is defined inductively as follows.

$$
\begin{aligned}
FV(1) &= FV(\mathrm{unit}) = \emptyset \\
FV(T(M_1, \ldots, M_n)) &= FV(f(M_1, \ldots, M_n)) = FV(M_1) \cup \cdots \cup FV(M_n) \\
FV(\Sigma x\colon A.B) &= FV(\Pi x\colon A.B) \;=\; FV(\Pi^* x\colon A.B) \;=\; FV(A) \cup (FV(B) \setminus \{x\}) \\
FV(x) &= \{x\} \\
FV(\mathrm{pair}_{(x:A)B}(M,N)) &= FV(A) \cup (FV(B) \setminus \{x\}) \cup FV(M) \cup FV(N) \\
FV(\mathrm{fst}_{(x:A)B}(M)) &= FV(A) \cup (FV(B) \setminus \{x\}) \cup FV(M) \\
FV(\mathrm{snd}_{(x:A)B}(M)) &= FV(A) \cup (FV(B) \setminus \{x\}) \cup FV(M) \\
FV(\lambda x\colon A.M) &= FV(A) \cup (FV(M) \setminus \{x\}) \\
FV(\mathrm{app}_{(x:A)B}(M,N)) &= FV(A) \cup (FV(B) \setminus \{x\}) \cup FV(M) \cup FV(N) \\
FV(\lambda^* x\colon A.M) &= FV(A) \cup (FV(M) \setminus \{x\}) \\
FV(\mathrm{app}^*_{(x:A)B}(M,N)) &= FV(A) \cup (FV(B) \setminus \{x\}) \cup FV(M) \cup FV(N)
\end{aligned}
$$

All variables that are not free are *bound variables*. We identify types and terms up to renaming of bound variables. We write $\Gamma[M/x], A[M/x]$ and $N[M/x]$ for the usual capture-avoiding substitution.

When the variable x is not free in the type B, we write $(A \times B)$ for $(\Sigma x\colon A.B)$ and $(A \to B)$ for $(\Pi x\colon A.B)$ and $(A \mathrel{-\!\!*} B)$ for $(\Pi^* x\colon A.B)$.

The type annotations on the terms can sometimes be hard to work with. In order to make terms more readable, we will frequently omit type annotations on terms. Nevertheless, this is just notational convenience: we assume the annotations to be there, even though we do not show them. We strip the type annotations as follows.

$$\begin{aligned} \mathit{strip}(\mathrm{pair}_{(x:A)B}(M,N)) &= \langle M,N\rangle \\ \mathit{strip}(\mathrm{fst}_{(x:A)B}(M)) &= \mathrm{fst}(M) \\ \mathit{strip}(\mathrm{snd}_{(x:A)B}(M)) &= \mathrm{snd}(M) \\ \mathit{strip}(\mathrm{app}_{(x:A)B}(M,N)) &= M\ N \\ \mathit{strip}(\mathrm{app}^*_{(x:A)B}(M,N)) &= M@N \end{aligned}$$

4.1.2 Judgements

The type theory has six kinds of judgements.

$\vdash \Gamma$ Bunch	The context Γ is well-formed.
$\Gamma \vdash A$ Type	The type A is well-formed in context Γ.
$\Gamma \vdash M : A$	The term M has type A in context Γ.
$\vdash \Gamma = \Delta$ Bunch	The contexts Γ and Δ are equal.
$\Gamma \vdash A = B$ Type	The types A and B in context Γ are equal.
$\Gamma \vdash M = N : A$	The terms M and N of type A in context Γ are equal.

We write $\Gamma \vdash \mathscr{J}$ to range over any of the judgements $(\Gamma \vdash A \text{ Type})$, $(\Gamma \vdash M : A)$, $(\Gamma \vdash A = B \text{ Type})$ and $(\Gamma \vdash M = N : A)$. We call a type A *closed* if the judgement $\vdash A$ Type is derivable and we call a term M of type A *closed* if the judgement $\vdash M : A$ is derivable. The judgements for equality define *definitional equality*.

The derivability of judgements is defined by mutual recursion. The inference rules are given below. In order to be able to derive non-trivial judgements, there are rules for introducing type and term constants as well as equational axioms. These constants and axioms are declared in a theory $\mathscr{T}$.

4.1.3 Dependently Typed Algebraic Theories

The natural theories to consider for a first-order dependent type theory are generalised algebraic theories [18], also called dependently typed algebraic theories in [85]. These theories generalise many-sorted equational theories.

The constants and axioms defined in a dependently typed algebraic theory are typed. In the formulation of a theory $\mathscr{T}$ we want to allow only well-formed types. Because the inference rules for typing

judgements ($\Gamma \vdash A$ Type) are defined by mutual induction with the rules for all the other statements, and the theory $\mathscr{T}$ itself provides the constants and axioms for the inference rules, we cannot define the theory $\mathscr{T}$ independently of the derivation of judgements. To avoid this circularity, we allow possibly non-wellformed judgements in a dependently typed algebraic theory, but formulate the rules of inference so that only well-formed judgements can ever be used. This is the approach taken in [85, 106].

Definition 4.1.1. A *dependently typed algebraic theory* $\mathscr{T}$ is given by the following data.

- For each context Γ a collection $\mathscr{T}(\Gamma)$ of type constants, such that $T \in \mathscr{T}(\Gamma)$ and $T \in \mathscr{T}(\Delta)$ implies $\Gamma = \Delta$.
- For each context Γ and each term A a collection $\mathscr{T}(\Gamma;A)$ of term constants, such that $f \in \mathscr{T}(\Gamma;A)$ and $f \in \mathscr{T}(\Delta;B)$ implies $\Gamma = \Delta$ and $A = B$.
- For each context Γ a collection $\mathscr{T}^{\mathscr{E}}(\Gamma)$ of type equality axioms $\Gamma \vdash A = B$ Type.
- For each context Γ and each term A a collection $\mathscr{T}^{\mathscr{E}}(\Gamma;A)$ of term equality axioms $\Gamma \vdash M = N : A$.

This definition is in fact more general than the rules for constants in the next section. The rules are formulated only for a context with a single variable $(x\colon A)$. We make this restriction for simplicity. It allows us to formulate the type theory without having to give a syntax for simultaneous substitution at the same time. Note, however, that using Σ-types we can encode constants in contexts of the form $(x_1 : A_1, \ldots, x_n : A_n)$. With $*$-types, introduced in the next chapter, we can encode constants in arbitrary bunched contexts, so that the restriction is not essential.

4.1.4 Rules of Inference

In the structural rules we use double lines for rules that can be used both from top to bottom and also from bottom to top.

We emphasise our convention that no variable may occur more than once in a context Γ. This convention adds implicit side-conditions to the rules. For example, the rule (WEAK) is subject to the condition that the variable x is not declared in Γ or Δ.

Rules marked with † will be discussed in more detail in Section 4.3.

4.1.4.1 Bunches†

$$(\textsc{Bu-Empty})\ \frac{}{\vdash \diamond \text{ Bunch}}$$

$$(\textsc{Bu-Add})\ \frac{\Gamma \vdash A \text{ Type}}{\vdash \Gamma, x\colon A \text{ Bunch}}$$

$$(\textsc{Bu-Mult})\ \frac{\vdash \Gamma \text{ Bunch} \qquad \vdash \Delta \text{ Bunch}}{\vdash \Gamma * \Delta \text{ Bunch}}$$

4.1.4.2 Structural rules†

$$(\textsc{Proj})\ \frac{\Gamma \vdash A\ \mathrm{Type}}{\Gamma, x\colon A \vdash x : A}$$

$$(\textsc{Weak})\ \frac{\Gamma(\Delta) \vdash \mathscr{J} \qquad \Delta \vdash A\ \mathrm{Type}}{\Gamma(\Delta, x\colon A) \vdash \mathscr{J}}$$

$$(\textsc{Unit})\ \frac{\Gamma(\Delta) \vdash \mathscr{J}}{\Gamma(\Delta * \diamond) \vdash \mathscr{J}} \qquad (\textsc{Swap})\ \frac{\Gamma(\Delta * \Phi) \vdash \mathscr{J}}{\Gamma(\Phi * \Delta) \vdash \mathscr{J}} \qquad (\textsc{Assoc})\ \frac{\Gamma((\Delta * \Phi) * \Psi) \vdash \mathscr{J}}{\Gamma(\Delta * (\Phi * \Psi)) \vdash \mathscr{J}}$$

4.1.4.3 Substitution

$$(\textsc{Subst})\ \frac{\Gamma(\Delta, x\colon A) \vdash \mathscr{J} \qquad \Delta \vdash M : A}{\Gamma(\Delta)\,[M/x] \vdash \mathscr{J}\,[M/x]}$$

$$(\textsc{Subst-Ty-Cgr})\ \frac{\Gamma(\Delta, x\colon A) \vdash B\ \mathrm{Type} \qquad \Delta \vdash M = N : A}{\Gamma(\Delta)\,[M/x] \vdash B\,[M/x] = B\,[N/x]\ \mathrm{Type}}$$

$$(\textsc{Subst-Tm-Cgr})\ \frac{\Gamma(\Delta, x\colon A) \vdash R : B \qquad \Delta \vdash M = N : A}{\Gamma(\Delta)\,[M/x] \vdash R\,[M/x] = R\,[N/x] : B\,[M/x]}$$

4.1.4.4 Conversion rules

$$(\textsc{Bu-Conv})\ \frac{\vdash \Delta = \Gamma\ \mathrm{Bunch} \qquad \Gamma \vdash \mathscr{J}}{\Delta \vdash \mathscr{J}}$$

$$(\textsc{Ty-Conv})\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash A = B\ \mathrm{Type}}{\Gamma \vdash M : B}$$

$$(\textsc{Ty-Eq-Conv})\ \frac{\Gamma \vdash M = N : A \qquad \Gamma \vdash A = B\ \mathrm{Type}}{\Gamma \vdash M = N : B}$$

4.1.4.5 Equations for bunches

$$(\textsc{Bu-Refl})\ \frac{\vdash \Gamma\ \mathrm{Bunch}}{\vdash \Gamma = \Gamma\ \mathrm{Bunch}}$$

$$(\textsc{Bu-Sym})\ \frac{\vdash \Gamma = \Delta\ \mathrm{Bunch}}{\vdash \Delta = \Gamma\ \mathrm{Bunch}}$$

$$(\textsc{Bu-Trans})\ \frac{\vdash \Gamma = \Delta\ \mathrm{Bunch} \qquad \vdash \Delta = \Phi\ \mathrm{Bunch}}{\vdash \Gamma = \Phi\ \mathrm{Bunch}}$$

$$(\textsc{Bu-AEq})\ \frac{\vdash \Gamma = \Delta\ \mathrm{Bunch} \qquad \Gamma \vdash A = B\ \mathrm{Type}}{\vdash (\Gamma, x\colon A) = (\Delta, x\colon B)\ \mathrm{Bunch}}$$

$$(\textsc{Bu-MEq})\ \frac{\vdash \Gamma = \Delta\ \mathrm{Bunch} \qquad \vdash \Phi = \Psi\ \mathrm{Bunch}}{\vdash \Gamma * \Phi = \Delta * \Psi\ \mathrm{Bunch}}$$

4.1.4.6 General equations for types

$$(\textsc{Ty-Refl})\ \frac{\Gamma \vdash A \text{ Type}}{\Gamma \vdash A = A \text{ Type}}$$

$$(\textsc{Ty-Sym})\ \frac{\Gamma \vdash A = B \text{ Type}}{\Gamma \vdash B = A \text{ Type}}$$

$$(\textsc{Ty-Trans})\ \frac{\Gamma \vdash A = B \text{ Type} \qquad \Gamma \vdash B = C \text{ Type}}{\Gamma \vdash A = C \text{ Type}}$$

4.1.4.7 General equations for terms

$$(\textsc{Tm-Refl})\ \frac{\Gamma \vdash M : A}{\Gamma \vdash M = M : A}$$

$$(\textsc{Tm-Sym})\ \frac{\Gamma \vdash M = N : A}{\Gamma \vdash N = M : A}$$

$$(\textsc{Tm-Trans})\ \frac{\Gamma \vdash M = N : A \qquad \Gamma \vdash N = P : A}{\Gamma \vdash M = P : A}$$

4.1.4.8 Rules for constants and axioms

$$(\textsc{C-Ty})\ \frac{\Gamma \vdash M : A \qquad \vdash x\colon A \text{ Bunch}}{\Gamma \vdash T(M) \text{ Type}}\ T \in \mathscr{T}(x\colon A)$$

$$(\textsc{C-Tm})\ \frac{\Gamma \vdash M : A \qquad x\colon A \vdash B \text{ Type}}{\Gamma \vdash f(M) : B[M/x]}\ f \in \mathscr{T}(x\colon A; B)$$

Congruences

$$(\textsc{C-Ty-Cgr})\ \frac{\Gamma \vdash M = N : A \qquad \vdash x\colon A \text{ Bunch}}{\Gamma \vdash T(M) = T(N) \text{ Type}}\ T \in \mathscr{T}(x\colon A)$$

$$(\textsc{C-Tm-Cgr})\ \frac{\Gamma \vdash M = N : A \qquad x\colon A \vdash B \text{ Type}}{\Gamma \vdash f(M) = f(N) : B[M/x]}\ f \in \mathscr{T}(x\colon A; B)$$

Axioms

$$(\textsc{Ty-Eq-Ax})\ \frac{\Gamma \vdash M : A \qquad x\colon A \vdash B \text{ Type} \qquad x\colon A \vdash C \text{ Type}}{\Gamma \vdash B[M/x] = C[M/x] \text{ Type}}\ (x\colon A \vdash B = C \text{ Type}) \in \mathscr{T}^{\mathscr{E}}(x\colon A)$$

$$(\textsc{Tm-Eq-Ax})\ \frac{\Gamma \vdash M : A \qquad x\colon A \vdash N : B \qquad x\colon A \vdash P : B}{\Gamma \vdash N[M/x] = P[M/x] : B[M/x]}\ (x\colon A \vdash N = P : B) \in \mathscr{T}^{\mathscr{E}}(x\colon A; B)$$

4.1.4.9 Rules for 1-types

Formation

$$(1\text{-TY})\ \frac{\vdash \Gamma\ \text{Bunch}}{\Gamma \vdash 1\ \text{Type}}$$

Introduction

$$(1\text{-I})\ \frac{\vdash \Gamma\ \text{Bunch}}{\Gamma \vdash \text{unit} : 1}$$

Equations

$$(1\text{-EQ})\ \frac{\Gamma \vdash M : 1}{\Gamma \vdash M = \text{unit} : 1}$$

4.1.4.10 Rules for Π-types

Formation

$$(\Pi\text{-TY})\ \frac{\Gamma \vdash A\ \text{Type} \qquad \Gamma, x\colon A \vdash B\ \text{Type}}{\Gamma \vdash \Pi x\colon A.B\ \text{Type}}$$

Introduction

$$(\Pi\text{-I})\ \frac{\Gamma, x\colon A \vdash M : B}{\Gamma \vdash \lambda x\colon A.M : \Pi x\colon A.B}$$

Elimination

$$(\Pi\text{-E})\ \frac{\Gamma, x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : \Pi x\colon A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash \text{app}_{(x:A)B}(M,N) : B[N/x]}$$

Congruences

$$(\Pi\text{-TY-CGR})\ \frac{\Gamma \vdash A_1 = A_2\ \text{Type} \qquad \Gamma, x\colon A_1 \vdash B_1 = B_2\ \text{Type}}{\Gamma \vdash \Pi x\colon A_1.B_1 = \Pi x\colon A_2.B_2\ \text{Type}}$$

$$(\Pi\text{-I-CGR})\ \frac{\Gamma \vdash A_1 = A_2\ \text{Type} \qquad \Gamma, x\colon A_1 \vdash M_1 = M_2 : B}{\Gamma \vdash \lambda x\colon A_1.M_1 = \lambda x\colon A_2.M_2 : \Pi x\colon A_1.B}$$

$$(\Pi\text{-E-CGR})\ \frac{\begin{array}{rr}\Gamma \vdash A_1 = A_2\ \text{Type} & \Gamma \vdash M_1 = M_2 : \Pi x\colon A_1.B_1 \\ \Gamma, x\colon A_1 \vdash B_1 = B_2\ \text{Type} & \Gamma \vdash N_1 = N_2 : A_1\end{array}}{\Gamma \vdash \text{app}_{(x:A_1)B_1}(M_1,N_1) = \text{app}_{(x:A_2)B_2}(M_2,N_2) : B_1[N_1/x]}$$

Equations

$$(\Pi\text{-}\beta)\ \frac{\Gamma, x\colon A \vdash M : B \qquad \Gamma \vdash N : A}{\Gamma \vdash \text{app}_{(x:A)B}((\lambda x\colon A.M),N) = M[N/x] : B[N/x]}$$

$$(\Pi\text{-}\eta)\ \frac{\Gamma, x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : \Pi x\colon A.B}{\Gamma \vdash \lambda x\colon A.(\text{app}_{(x:A)B}(M,x)) = M : \Pi x\colon A.B}$$

4.1.4.11 Rules for Σ-types

Formation

$$(\Sigma\text{-TY})\ \frac{\Gamma \vdash A\ \text{Type} \qquad \Gamma, x\colon A \vdash B\ \text{Type}}{\Gamma \vdash \Sigma x\colon A.B\ \text{Type}}$$

Introduction

$$(\Sigma\text{-I})\ \frac{\Gamma, x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : B[M/x]}{\Gamma \vdash \text{pair}_{(x:A)B}(M,N) : \Sigma x\colon A.B}$$

Elimination

$$(\Sigma\text{-E1})\ \frac{\Gamma, x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : \Sigma x\colon A.B}{\Gamma \vdash \text{fst}_{(x:A)B}(M) : A}$$

$$(\Sigma\text{-E2})\ \frac{\Gamma, x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : \Sigma x\colon A.B}{\Gamma \vdash \text{snd}_{(x:A)B}(M) : B[\text{fst}_{(x:A)B}(M)/x]}$$

Congruences

$$(\Sigma\text{-TY-CGR})\ \frac{\Gamma \vdash A_1 = A_2\ \text{Type} \qquad \Gamma, x\colon A_1 \vdash B_1 = B_2\ \text{Type}}{\Gamma \vdash \Sigma x\colon A_1.B_1 = \Sigma x\colon A_2.B_2\ \text{Type}}$$

$$(\Sigma\text{-I-CGR})\ \frac{\begin{array}{cc}\Gamma \vdash A_1 = A_2\ \text{Type} & \Gamma \vdash M_1 = M_2 : A_1 \\ \Gamma, x\colon A_1 \vdash B_1 = B_2\ \text{Type} & \Gamma \vdash N_1 = N_2 : B_1[M_1/x]\end{array}}{\Gamma \vdash \text{pair}_{(x:A_1)B_1}(M_1,N_1) = \text{pair}_{(x:A_2)B_2}(M_2,N_2) : \Sigma x\colon A_1.B_1}$$

$$(\Sigma\text{-E1-CGR})\ \frac{\begin{array}{cc}\Gamma \vdash A_1 = A_2\ \text{Type} & \\ \Gamma, x\colon A_1 \vdash B_1 = B_2\ \text{Type} & \Gamma \vdash M_1 = M_2 : \Sigma x\colon A_1.B_1\end{array}}{\Gamma \vdash \text{fst}_{(x:A_1)B_1}(M_1) = \text{fst}_{(x:A_2)B_2}(M_2) : A_1}$$

$$(\Sigma\text{-E2-CGR})\ \frac{\begin{array}{cc}\Gamma \vdash A_1 = A_2\ \text{Type} & \\ \Gamma, x\colon A_1 \vdash B_1 = B_2\ \text{Type} & \Gamma \vdash M_1 = M_2 : \Sigma x\colon A_1.B_1\end{array}}{\Gamma \vdash \text{snd}_{(x:A_1)B_1}(M_1) = \text{snd}_{(x:A_2)B_2}(M_2) : B_1[\text{fst}_{(x:A_1)B_1}(M_1)/x]}$$

Equations

$$(\Sigma\text{-}\beta 1)\ \frac{\Gamma, x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : B[M/x]}{\Gamma \vdash \text{fst}_{(x:A)B}(\text{pair}_{(x:A)B}(M,N)) = M : A}$$

$$(\Sigma\text{-}\beta 2)\ \frac{\Gamma, x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : B[M/x]}{\Gamma \vdash \text{snd}_{(x:A)B}(\text{pair}_{(x:A)B}(M,N)) = N : B[M/x]}$$

$$(\Sigma\text{-}\eta)\ \frac{\Gamma, x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : \Sigma x\colon A.B}{\Gamma \vdash M = \text{pair}_{(x:A)B}(\text{fst}_{(x:A)B}(M), \text{snd}_{(x:A)B}(M)) : \Sigma x\colon A.B}$$

4.1.4.12 Rules for strong Π^*-types†

Formation

$$(\Pi^*\text{-TY})\ \frac{\vdash A\ \text{Type} \qquad \Gamma * x\colon A \vdash B\ \text{Type}}{\Gamma \vdash \Pi^* x\colon A.B\ \text{Type}}$$

Introduction

$$(\Pi^*\text{-I})\ \frac{\Gamma * x\colon A \vdash M : B}{\Gamma \vdash \lambda^* x\colon A.M : \Pi^* x\colon A.B}$$

Elimination

$$(\Pi^*\text{-E})\ \frac{\Gamma * x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : \Pi^* x\colon A.B \qquad \Delta \vdash N : A}{\Gamma * \Delta \vdash \text{app}^*_{(x:A)B}(M,N) : B[N/x]}$$

Congruences

$$(\Pi^*\text{-TY-CGR})\ \frac{\vdash A_1 = A_2\ \text{Type} \qquad \Gamma * x\colon A_1 \vdash B_1 = B_2\ \text{Type}}{\Gamma \vdash \Pi^* x\colon A_1.B_1 = \Pi^* x\colon A_2.B_2\ \text{Type}}$$

$$(\Pi^*\text{-I-CGR})\ \frac{\vdash A_1 = A_2\ \text{Type} \qquad \Gamma * x\colon A_1 \vdash M_1 = M_2 : B}{\Gamma \vdash \lambda^* x\colon A_1.M_1 = \lambda^* x\colon A_2.M_2 : \Pi^* x\colon A_1.B}$$

$$(\Pi^*\text{-E-CGR})^\dagger\ \frac{\begin{array}{ll} & \vdash A_1 = A_2\ \text{Type} \\ & \Gamma \vdash \Pi^* x\colon A_1.B_1 = \Pi^* x\colon A_2.B_2\ \text{Type} \\ & \Gamma \vdash B_1[N_1/x] = B_2[N_2/x]\ \text{Type} \\ \Gamma \vdash \text{app}^*_{(x:A_1)B_1}(M_1,N_1) : B_1[N_1/x] & \Gamma \vdash M_1 = M_2 : \Pi^* x\colon A_1.B_1 \\ \Gamma \vdash \text{app}^*_{(x:A_2)B_2}(M_2,N_2) : B_2[N_2/x] & \Gamma \vdash N_1 = N_2 : A_1 \end{array}}{\Gamma \vdash \text{app}^*_{(x:A_1)B_1}(M_1,N_1) = \text{app}^*_{(x:A_2)B_2}(M_2,N_2) : B_1[N_1/x]}$$

Equations

$$(\Pi^*\text{-}\beta)\ \frac{\Gamma * x\colon A \vdash M : B \qquad \Delta \vdash N : A}{\Gamma * \Delta \vdash \text{app}^*_{(x:A)B}((\lambda^* x\colon A.M),N) = M[N/x] : B[N/x]}$$

$$(\Pi^*\text{-}\eta)\ \frac{\Gamma * x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : \Pi^* x\colon A.B}{\Gamma \vdash \lambda^* x\colon A.(\text{app}^*_{(x:A)B}(M,x)) = M : \Pi^* x\colon A.B}$$

4.2 Example

We give a simple example to demonstrate the use of the rules for bunches and the structural rules. We show that, given an element of type $\Pi x\colon A.B$, we can obtain an element of type $\Pi^* x\colon A.B$. If we think of Π^* as a type of partial functions then this expresses that any total function can be restricted to a partial function.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x\colon A \vdash B\ \mathrm{Type}}{\vdash \Pi x\colon A.B\ \mathrm{Type}}\,(\Pi\text{-Ty})}{f\colon \Pi x\colon A.B \vdash f : \Pi x\colon A.B}\,(\textsc{Proj})}{(f\colon \Pi x\colon A.B) * \diamond \vdash f : \Pi x\colon A.B}\,(\textsc{Unit})}{(f\colon \Pi x\colon A.B) * x\colon A \vdash f : \Pi x\colon A.B}\,(\textsc{Weak}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\vdash A\ \mathrm{Type}}{x\colon A \vdash x : A}\,(\textsc{Proj})}{x\colon A * \diamond \vdash x : A}\,(\textsc{Unit})}{x\colon A * (f\colon \Pi x\colon A.B) \vdash x : A}\,(\textsc{Weak})}{(f\colon \Pi x\colon A.B) * x\colon A \vdash x : A}\,(\textsc{Swap})}{(f\colon \Pi x\colon A.B) * x\colon A \vdash \mathrm{app}_{(x:A)B}(f,x) : B}\,(\Pi\text{-E})$$

$$\dfrac{(f\colon \Pi x\colon A.B) * x\colon A \vdash \mathrm{app}_{(x:A)B}(f,x) : B}{f\colon \Pi x\colon A.B \vdash \lambda^* x\colon A.\mathrm{app}_{(x:A)B}(f,x) : \Pi^* x\colon A.B}\,(\Pi^*\text{-I})$$

This derivation relies on the rule (Unit) by which the empty context is a unit for $*$. In the left branch of the derivation, (Unit) is used in conjunction with (Weak) to obtain a multiplicative weakening from the context $(f\colon \Pi x\colon A.B)$ to the context $(f\colon \Pi x\colon A.B) * x\colon A$. More generally, we have the following principle of multiplicative weakening.

Lemma 4.2.1. *The following rule is admissible.*

$$(*\text{-Weak})\ \dfrac{\vdash \Phi\ \mathrm{Bunch} \qquad \Gamma(\Delta) \vdash \mathscr{J}}{\Gamma(\Delta * \Phi) \vdash \mathscr{J}}$$

Proof. By induction on the structure of Φ, using (Unit) and (Weak). □

As a consequence of this lemma we get arbitrary weakening.

Lemma 4.2.2. *The following rule is admissible.*

$$(\mathrm{GWeak})\ \dfrac{\Gamma(\Delta) \vdash \mathscr{J} \qquad \vdash \Phi(\Delta)\ \mathrm{Bunch}}{\Gamma(\Phi(\Delta)) \vdash \mathscr{J}}$$

Proof. By induction on $\Phi_\circ$, using (Weak) and the previous lemma. □

4.3 Discussion

The rules for the additive types such as Π and Σ are completely standard. The rules for Π, for example, differ from those in [105] only in that in the rule $(\Pi\text{-}\beta)$ we omit the assumption $\Gamma, x\colon A \vdash B\ \mathrm{Type}$, which can be inferred from $\Gamma, x\colon A \vdash M : B$.

Other additive type formers such as binary sum types and identity types can be added to the type theory using their well-known formulation. As an example consider the elimination rule for binary coproducts. This rule is usually formulated as follows, see e.g. [81], where, for simplicity, we only show the special case of the rule where z is not substituted for.

$$(+\text{-E1})\ \frac{\Gamma, z\colon A+B \vdash C \text{ Type} \qquad \begin{array}{c}\Gamma, x\colon A \vdash M : C[\kappa_1(x)/z] \\ \Gamma, y\colon B \vdash N : C[\kappa_2(x)/z]\end{array}}{\Gamma, z\colon A+B \vdash \text{case } z \text{ of } (\kappa_1(x).M,\ \kappa_2(y).N) : C}$$

This rule makes the assumption that the variable z occurs at the end of the context, i.e. there can be no parameter context following z. In practice, however, one would like to use a rule with parameter contexts, such as the following.

$$(+\text{-E2})\ \frac{\Gamma(z\colon A+B) \vdash C \text{ Type} \qquad \begin{array}{c}\Gamma(x\colon A)[\kappa_1(x)/z] \vdash M : C[\kappa_1(x)/z] \\ \Gamma(y\colon B)[\kappa_1(x)/z] \vdash N : C[\kappa_2(x)/z]\end{array}}{\Gamma(z\colon A+B) \vdash \text{case } z \text{ of } (\kappa_1(x).M,\ \kappa_2(y).N) : C}$$

In the usual additive dependent type theory with Π-types, the second rule can be derived from the first rule. For example, suppose the context $\Gamma(z\colon A+B)$ is $z\colon A+B, u\colon D$. If we write C_1 for $C[\kappa_1(x)/z]$, C_2 for $C[\kappa_2(y)/z]$, D_1 for $D[\kappa_1(x)/z]$ and D_2 for $D[\kappa_2(y)/z]$, then the first rule can be applied to the sequents $x\colon A \vdash \lambda u\colon D_1.M : \Pi u\colon D_1.C_1$ and $y\colon B \vdash \lambda u\colon D_2.N : \Pi u\colon D_2.C_2$ to derive

$$z\colon A+B \vdash \text{case } z \text{ of } (\kappa_1(x).\lambda u\colon D_1.M,\ \kappa_2(y).\lambda u\colon D_2.N) : \Pi u\colon D.C.$$

Using weakening and application we can therefore get

$$z\colon A+B, u\colon D \vdash \text{app}_{(u:D)C}(\text{case } z \text{ of } (\kappa_1(x).\lambda u\colon D_1.M,\ \kappa_2(y).\lambda u\colon D_2.N), u) : C,$$

as required for the second rule. Note, however, that this derivation gives us a more elaborate term than the second rule.

In $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$ we cannot always derive the second rule from the first rule. If the context $\Gamma(z\colon A+B)$ has the form $\Gamma'(z\colon A+B) * u\colon D$, then it is possible to use Π^*-types in a similar way as Π-types above to get to the context $\Gamma'(z\colon A+B)$. But if the context $\Gamma(z\colon A+B)$ is $(\Gamma, z\colon A+B) * ((t\colon D * u\colon E), v\colon F)$, then there is no way in which we could use Π-types and Π^*-types to eliminate the parameter context.

We remove this defect in the next chapter by introducing $*$-types, using which we can transform the context $(\Gamma, z\colon A+B) * ((t\colon D * u\colon E), v\colon F)$ into a context of the form $(\Gamma, z\colon A+B) * (p\colon (\Sigma q\colon D * E.F))$ for which we can use Π^*-types as described above. Another possibility of removing this defect is to restrict the bunches such that $\Gamma * x\colon A$ is the only allowed form of bunch multiplication. Then, clearly, Π and Π^* suffice to eliminate the parameter contexts. For instance, in his formulation of Nominal Logic, Cheney [20] only needs such a restricted form of bunched contexts. Here we do not make this restriction as it would force us to restrict substitution for x in order to maintain this form of restricted context. It would however be interesting to consider such a restricted system, as it should be simpler than the present one.

At this point, it is fair to ask why we do not use the rule $(+\text{-E2})$ with parameter contexts in the first place. The main reason for preferring $(+\text{-E1})$ is that it allows fewer derivations of the same term

than (+-E2). Without parameter contexts, the position of the variable z in the context is determined by the rule. In the rule with parameter contexts, however, there may be many positions in the context where z could occur, which means that there may be many derivations of the same term. That the rule with parameter contexts allows more derivation makes the semantic interpretation more complicated. To prove soundness of the interpretation of the syntax, we have to show that two derivations of the same sequent have the same interpretation, and this is more difficult the more derivations of the sequent there are. In fact, we are not aware of any such proof of soundness for a rule with parameter contexts. Notice that in the above derivation of (+-E2) from (+-E1), the terms contain additional information that allows us to distinguish different derivations that would receive the same term in the rule (+-E2). For example, if, in the above derivation of (+-E2) from (+-E1), we consider for the context $\Gamma(z\colon A+B)$ the cases $(z\colon A+B, u\colon D)$ and $(u\colon D, z\colon A+B)$, then we obtain different terms in each case, while the rule (+-E2) gives us the same term in both cases.

Coming to the rules for bunches in Section 4.1.4.1, we remark that the present rules make bunches asymmetric. For example, we can form a bunch of the form $((\Gamma * \Delta), x\colon A)$ but we *cannot* form a bunch of the form $(x\colon A, (\Gamma * \Delta))$. Indeed, our semantics does not explain what it would mean to multiply two contexts in presence of the assumption $x\colon A$, i.e. it does not explain $x\colon A \vdash \Gamma * \Delta$, as would be necessary to form $(x\colon A, (\Gamma * \Delta))$. The asymmetry of context concatenation is also present in normal dependent type theory. There, contexts are just lists of assumptions, which means that they are generated from the empty context by the extension that takes Γ to $\Gamma, x\colon A$. The notation $(x\colon A, (\Gamma, \Delta))$ is just a short-hand for the concatenation of lists. To make available sequents of the form $(x\colon A, (\Gamma * \Delta))$ in the bunched type theory, it is possible to add a cartesian product of contexts. Then we could write $((x\colon A) \times (\Gamma * \Delta))$. This is not necessary here, since in the next chapter we introduce $*$-types. Using $*$-types and Σ-types, we can internalise contexts as types. For example, the contexts $(y\colon B, z\colon C) * (u\colon D)$ and $w\colon (\Sigma y\colon B.C) * D$ are isomorphic. By internalising $\Gamma * \Delta$ we can therefore get essentially the context $((x\colon A) \times (\Gamma * \Delta))$.

The structural rules in Section 4.1.4.2 make $*$ a symmetric monoidal structure. Moreover, by rule (UNIT) the empty context $\diamond$ is a unit of the monoidal structure $*$. This makes the monoidal structure $*$ affine. Apart from the fact that affineness is desired when $*$ is a monoidal structure capturing freshness, there is also a technical reason for assuming $*$ to be affine. It has to do with proving soundness of the categorical semantics of the type theory and is sketched in Section 6.2.

As is the case for O'Hearn and Pym's $\alpha\lambda$-calculus, the structural rules are not admissible. The following judgement, taken from [80], is derivable, but not without weakening.

$$((f\colon A \mathrel{-\!\!*} B) * (x\colon A)), z\colon C \vdash \mathrm{app}^{*}_{(x:A)B}(f,x) : B$$

As argued in [80], this example also shows that it is not enough to build the structural rules into the projection rule, as is usually done in non-bunched type theory. Even if we replace (PROJ) with the (admissible) rule

$$\frac{\vdash \Gamma(\Delta)\ \mathrm{Bunch} \qquad \Delta \vdash A\ \mathrm{Type}}{\Gamma(\Delta, x\colon A) \vdash x : A},$$

the above sequent still requires an application of weakening after (Π^*-E).

The rules for Π^*-types are similar to the rules for $-\!\!*$ in O'Hearn and Pym's $\alpha\lambda$-calculus. In the rule (Π^*-TY), the type A must be a closed type, since $(\vdash \Gamma * x\colon A \text{ Bunch})$ is derivable only for a closed type A. This corresponds to the fact that Π^*-types represent *simple* monoidal products. Of the other rules for Π^*, the congruence rule (Π^*-E-Cgr) requires some comment. Perhaps one would have expected the following rule instead.

$$(\Pi^*\text{-E-CGR}') \quad \frac{\begin{array}{rr} \vdash A_1 = A_2 \text{ Type} & \Gamma \vdash M_1 = M_2 : \Pi^* x\colon A_1.B_1 \\ \Gamma * x\colon A_1 \vdash B_1 = B_2 \text{ Type} & \Delta \vdash N_1 = N_2 : A_1 \end{array}}{\Gamma * \Delta \vdash \mathrm{app}^*_{(x:A_1)B_1}(M_1,N_1) = \mathrm{app}^*_{(x:A_2)B_2}(M_2,N_2) : B_1[N_1/x]}$$

It is easy to show that (Π^*-E-CGR$'$) is admissible from the rule (Π^*-E-CGR). From the assumptions of (Π^*-E-CGR$'$), we have derivations

$$\begin{array}{r} (\Pi^*\text{-TY-CGR}) \\ \text{Lemma 4.2.2} \end{array} \frac{\Gamma * x\colon A_1 \vdash B_1 = B_2 \text{ Type}}{\Gamma \vdash \Pi^* x\colon A_1.B_1 = \Pi^* x\colon A_2.B_2 \text{ Type}}}{\Gamma * \Delta \vdash \Pi^* x\colon A_1.B_1 = \Pi^* x\colon A_2.B_2 \text{ Type}}$$

and

$$\begin{array}{r} \text{Lemma 4.2.2} \\ (\text{SUBST-TY-CGR}) \end{array} \frac{\dfrac{\Gamma * x\colon A_1 \vdash B_1 = B_2 \text{ Type}}{\Gamma * (\Delta, x\colon A_1) \vdash B_1 = B_2 \text{ Type}} \qquad \Delta \vdash N_1 = N_2 : A_1}{\Gamma * \Delta \vdash B_1[N_1/x] = B_2[N_2/x] \text{ Type}}$$

by means of which we get the necessary assumptions for (Π^*-E-CGR). The reason for choosing the rule (Π^*-E-CGR) over (Π^*-E-CGR$'$) is that in (Π^*-E-CGR) it suffices to show the equations $M_1 = M_2$ and $N_1 = N_2$ in the current context and not in some smaller context. This makes (Π^*-E-CGR) easier to use. Moreover, the rule (Π^*-E-CGR) is the justification for annotating the term $\mathrm{app}^*_{(x:A)B}(M,N)$ with the types A and B but not with information on the contexts in which M and N were originally formed, since these contexts are not relevant to the identity of $\mathrm{app}^*_{(x:A)B}(M,N)$. If we only have the rule (Π^*-E-CGR$'$), then to relate $\mathrm{app}^*_{(x:A_1)B_1}(M_1,N_1)$ and $\mathrm{app}^*_{(x:A_2)B_2}(M_2,N_2)$ we need information about the contexts in which M_i and N_i were formed. In general, there seems to be no way of relating these contexts without including them in the annotations for app*. This problem also appears in the semantic interpretation, where we need to assume strong Π^*-types if we do not want context annotations on app*-terms. The congruence rule (Π^*-E-CGR) corresponds to the assumption that the Π^*-types are strong. We conjecture that in a type theory with strengthening, such context annotations are not necessary.

4.4 Basic Properties

As a sanity check for the rules of the type theory, we check some of the basic properties that one expects of a type theory.

Lemma 4.4.1. *The following are true.*

1. *If* $\Gamma \vdash A$ Type *then* $FV(A) \subseteq FV(\Gamma)$.
2. *If* $\Gamma \vdash M : A$ *then* $FV(M) \cup FV(A) \subseteq FV(\Gamma)$.
3. *If* $\Gamma \vdash A = B$ Type *then* $FV(A) \cup FV(B) \subseteq FV(\Gamma)$.
4. *If* $\Gamma \vdash M = N : A$ *then* $FV(M) \cup FV(N) \cup FV(A) \subseteq FV(\Gamma)$.

Proof. By straightforward induction on derivations. □

Lemma 4.4.2. *Let* α *be an injective variable renaming. Then we have:*

1. *If* $\vdash \Gamma$ Bunch *then* $\vdash \Gamma[\alpha]$ Bunch.
2. *If* $\Gamma \vdash \mathscr{J}$ *then* $\Gamma[\alpha] \vdash \mathscr{J}[\alpha]$.

Proof. By straightforward induction on derivations, since all rules are closed under injective variable renamings. □

Lemma 4.4.3. *The following are true.*

1. *If* $\vdash \Gamma(\Delta)$ Bunch *then* $\vdash \Delta$ Bunch.
2. *If* $\vdash \Gamma(\Delta)$ Bunch *and* $\Delta \vdash A$ Type *then* $\vdash \Gamma(\Delta, x\colon A)$ Bunch.
3. *If* $\vdash \Gamma(\Delta, x\colon A)$ Bunch *and* $\Delta \vdash M : A$ *then* $\vdash \Gamma(\Delta)[M/x]$ Bunch.
4. *If* $\vdash \Gamma * \Delta$ Bunch *then* $\vdash \Delta * \Gamma$ Bunch
5. $\vdash (\Gamma * \Delta) * \Phi$ Bunch *if and only if* $\vdash \Gamma * (\Delta * \Phi)$ Bunch.

Proof. The first three points follow by induction on the size of $\Gamma_\circ$. The last two points follow using inversion. □

Lemma 4.4.4 (Type inversion)**.** *The following are true.*

1. *If* $\Gamma \vdash T(M)$ Type *then there exists A such that* $\Gamma \vdash M : A$ *and* $\vdash x\colon A$ Bunch *and* $T \in \mathscr{T}(x\colon A)$.
2. *If* $\Gamma \vdash A * B$ Type *then* $\vdash A$ Type *and* $\vdash B$ Type.
3. *If* $\Gamma \vdash \Pi x\colon A.B$ Type *then* $\Gamma \vdash A$ Type *and* $\Gamma, x\colon A \vdash B$ Type.
4. *If* $\Gamma \vdash \Sigma x\colon A.B$ Type *then* $\Gamma \vdash A$ Type *and* $\Gamma, x\colon A \vdash B$ Type.
5. *If* $\Gamma \vdash \Pi^* x\colon A.B$ Type *then* $\vdash A$ Type *and* $\Gamma * x\colon A \vdash B$ Type.

Proof. We consider 3, the other cases are similar. A derivation of the judgement $\Gamma \vdash \Pi x\colon A.B$ Type must end with an application of the rule (Π-TY) followed by a number of applications of the rules (BU-CONV), (WEAK), (SUBST), (UNIT), (SWAP) and (ASSOC). We continue by induction on the number of uses of these rules. If there are no applications of these rules then we are done, since the required judgements $\Gamma \vdash A$ Type and $\Gamma \vdash \Pi x\colon A.B$ Type are immediate premises of (Π-TY). For the induction case, we consider just the case where the derivation ends in an application of (BU-CONV) with the equality $\vdash \Delta = \Gamma$ Bunch. By induction hypothesis, we can derive $\Delta \vdash A$ Type and $\Delta, x\colon A \vdash B$ Type. By (BU-AEQ) we have $\vdash (\Delta, x\colon A) = (\Gamma, x\colon A)$ Bunch, so that we can use (Bu-Conv) to derive the required $\Gamma \vdash A$ Type and $\Gamma, x\colon A \vdash B$ Type. The other cases follow similarly. □

Lemma 4.4.5 (Syntactic Validity)**.** *The following are true.*

1. *If* $\Gamma \vdash \mathscr{J}$ *then* $\vdash \Gamma$ Bunch.
2. *If* $\Gamma \vdash A$ Type *then* $\vdash \Gamma$ Bunch.
3. *If* $\Gamma \vdash M : A$ *then* $\vdash \Gamma$ Bunch *and* $\Gamma \vdash A$ Type.
4. *If* $\vdash \Gamma = \Delta$ Bunch *then* $\vdash \Gamma$ Bunch *and* $\vdash \Delta$ Bunch.
5. *If* $\Gamma \vdash A = B$ Type *then* $\vdash \Gamma$ Bunch *and* $\Gamma \vdash A$ Type *and* $\Gamma \vdash B$ Type.
6. *If* $\Gamma \vdash M = N : A$ *then* $\vdash \Gamma$ Bunch *and* $\Gamma \vdash A$ Type *and* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : A$.

Proof. By simultaneous induction on derivations. We give some representative examples showing that all the rules from Section 4.1.4 preserve properties 1–6.

- Case (Π-E).

$$(\Pi\text{-E})\ \frac{\Gamma, x\colon A \vdash B \text{ Type} \qquad \Gamma \vdash M : \Pi x\colon A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathrm{app}_{(x:A)B}(M,N) : B[N/x]}$$

  We can derive the required $\Gamma \vdash B[N/x]$ Type using (SUBST).

- Case (Π-E-CGR).

$$(\Pi\text{-E-CGR})\ \frac{\begin{array}{rr}\Gamma \vdash A_1 = A_2 \text{ Type} & \Gamma \vdash M_1 = M_2 : \Pi x\colon A_1.B_1 \\ \Gamma, x\colon A_1 \vdash B_1 = B_2 \text{ Type} & \Gamma \vdash N_1 = N_2 : A_1\end{array}}{\Gamma \vdash \mathrm{app}_{(x:A_1)B_1}(M_1,N_1) = \mathrm{app}_{(x:A_2)B_2}(M_2,N_2) : B_1[N_1/x]}$$

  By induction hypothesis we have $\vdash \Gamma$ Bunch and $\Gamma, x\colon A_1 \vdash B_1$ Type and $\Gamma \vdash N_1 : A_1$ and $\Gamma \vdash M_1 : \Pi x\colon A_1.B_1$. By (SUBST) we get $\Gamma \vdash B_1[N_1/x]$ Type. By (Π-E) we get $\Gamma \vdash \mathrm{app}_{(x:A_1)B_1}(M_1,N_1) : B_1[N_1/x]$. Using the induction hypothesis we furthermore get $\Gamma, x\colon A_1 \vdash B_2$ Type, $\Gamma \vdash N_2 : A_1$ and $\Gamma \vdash M_2 : \Pi x\colon A_1.B_1$. Using (BU-AEQ) and (BU-CONV) we obtain $\Gamma, x\colon A_2 \vdash B_2$ Type. Using (Π-TY-CGR) we get $\Gamma \vdash \Pi x\colon A_1.B_1 = \Pi x\colon A_2.B_2$ Type. Using (TY-CONV) this implies

$\Gamma \vdash N_2 : A_2$ and $\Gamma \vdash M_2 : \Pi x\colon A_2.\, B_2$. By ($\Pi$-E) we get $\Gamma \vdash \mathrm{app}_{(x:A_2)B_2}(M_2, N_2) : B_2[N_2/x]$. Using (SUBST-TY-CGR) we get $\Gamma \vdash B_1[N_1/x] = B_2[N_2/x]$ Type. By symmetry and type conversion we finally get $\Gamma \vdash \mathrm{app}_{(x:A_2)B_2}(M_2, N_2) : B_1[N_1/x]$, thus completing the case.

□

For the formulation of properties of the type theory, we introduce a notion of subcontext up to structural congruence. This is given by the binary relation $\succcurlyeq$ between bunches defined by the following rules.

$$\frac{\vdash \Gamma \text{ Bunch}}{\Gamma \succcurlyeq \Gamma} \qquad \frac{\Phi \succcurlyeq \Delta \qquad \Delta \succcurlyeq \Gamma}{\Phi \succcurlyeq \Gamma}$$

$$\frac{\Delta \succcurlyeq \Gamma \qquad \Psi \succcurlyeq \Phi}{\Delta * \Psi \succcurlyeq \Gamma * \Phi} \qquad \frac{\Delta \succcurlyeq \Gamma \qquad \Delta \vdash A \text{ Type}}{(\Delta, x\colon A) \succcurlyeq (\Gamma, x\colon A)}$$

$$\frac{\Gamma \vdash A \text{ Type}}{(\Gamma, x\colon A) \succcurlyeq \Gamma}$$

$$\frac{\vdash \Gamma \text{ Bunch}}{\Gamma * \diamond \succcurlyeq \Gamma} \qquad \frac{\vdash \Gamma \text{ Bunch}}{\Gamma \succcurlyeq \Gamma * \diamond}$$

$$\frac{\vdash \Gamma \text{ Bunch} \qquad \vdash \Delta \text{ Bunch}}{\Delta * \Gamma \succcurlyeq \Gamma * \Delta}$$

$$\frac{\vdash \Gamma \text{ Bunch} \qquad \vdash \Delta \text{ Bunch} \qquad \vdash \Phi \text{ Bunch}}{\Gamma * (\Delta * \Phi) \succcurlyeq (\Gamma * \Delta) * \Phi}$$

$$\frac{\vdash \Gamma \text{ Bunch} \qquad \vdash \Delta \text{ Bunch} \qquad \vdash \Phi \text{ Bunch}}{(\Gamma * \Delta) * \Phi \succcurlyeq \Gamma * (\Delta * \Phi)}$$

We write $\Gamma \equiv \Delta$ for the conjunction of $\Gamma \succcurlyeq \Delta$ and $\Delta \succcurlyeq \Gamma$. Note that $\Gamma \equiv \Delta$ means that Γ and Δ are identical up to structural rearranging, while provable equality $\vdash \Gamma = \Delta$ Bunch means that, without structural rearrangement, Γ can be obtained from Δ by replacing types with provably equal ones.

Clearly, the following rule is equivalent to all of (UNIT), (SWAP), (WEAK) and (ASSOC).

$$\text{(CONG)}\ \frac{\Delta \vdash \mathscr{J} \qquad \Gamma \succcurlyeq \Delta}{\Gamma \vdash \mathscr{J}}$$

Lemma 4.4.6. *If $\Gamma \succcurlyeq \Delta(\Phi * \Psi)$ then there exist $\Delta'_\circ$, Φ' and Ψ' such that $\Phi' \succcurlyeq \Phi$ and $\Psi' \succcurlyeq \Psi$ and $\Gamma \equiv \Delta'(\Phi' * \Psi')$.*

Proof. By induction on the derivation of $\Gamma \succcurlyeq \Delta(\Phi * \Psi)$. □

Lemma 4.4.7. *If $\Gamma \succcurlyeq \Delta(\Phi, x\colon A)$ then there exist $\Delta'_\circ$ and Φ' such that $\Phi' \succcurlyeq \Phi$ and Γ is $\Delta'(\Phi', x\colon A)$.*

Proof. By induction on the derivation of $\Gamma \succcurlyeq \Delta(\Phi, x\colon A)$. □

Lemma 4.4.8. *If $\vdash \Gamma = \Delta$ Bunch and $\Gamma' \succcurlyeq \Gamma$ then there exists Δ' such that $\Delta' \succcurlyeq \Delta$ and $\vdash \Gamma' = \Delta'$ Bunch.*

Proof. By induction on the derivation of $\Gamma' \succcurlyeq \Gamma$. □

Lemma 4.4.9 (Term inversion)**.** *The following are true.*

1. *If* $\Gamma \vdash f(M)$ Type *then there exist* A *and* B *with* $\Gamma \vdash M : A$ *and* $x\colon A \vdash B$ Type *and* $f \in \mathscr{T}(x\colon A; B)$.
2. *If* $\Gamma \vdash \lambda x\colon A.M : C$ *then there exists* B *such that* $\Gamma \vdash C = \Pi x\colon A.B$ Type *and* $\Gamma, x\colon A \vdash M : B$.
3. *If* $\Gamma \vdash \mathrm{app}_{(x:A)B}(M,N) : C$ *then* $\Gamma \vdash C = \Pi x\colon A.B$ Type *and* $\Gamma \vdash M : \Pi x\colon A.B$ *and* $\Gamma \vdash N : A$.
4. *If* $\Gamma \vdash \mathrm{pair}_{(x:A)B}(M,N) : C$ *then* $\Gamma \vdash C = \Sigma x\colon A.B$ Type *and* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : B[M/x]$.
5. *If* $\Gamma \vdash \mathrm{fst}_{(x:A)B}(M) : C$ *then* $\Gamma \vdash C = A$ Type *and* $\Gamma \vdash M : \Sigma x\colon A.B$.
6. *If* $\Gamma \vdash \mathrm{snd}_{(x:A)B}(M) : C$ *then* $\Gamma \vdash C = B[\mathrm{fst}_{(x:A)B}(M)/x]$ Type *and* $\Gamma \vdash M : \Sigma x\colon A.B$.
7. *If* $\Gamma \vdash \lambda^* x\colon A.M : C$ *then there exists* B *such that* $\Gamma \vdash C = \Pi^* x\colon A.B$ Type *and* $\Gamma * x\colon A \vdash M : B$.
8. *If* $\Gamma \vdash \mathrm{app}^*_{(x:A)B}(M,N) : C$ *then there exist* $\Delta_\circ$, Φ *and* Ψ *such that* $\Gamma \equiv \Delta(\Phi * \Psi)$ *and* $\Phi * x\colon A \vdash B$ Type *and* $\Gamma \vdash C = \Pi^* x\colon A.B$ Type *and* $\Phi \vdash M : \Pi^* x\colon A.B$ *and* $\Psi \vdash N : A$.

Proof. The cases for Π-types and Σ-types are standard. We consider the case for app*. Any derivation of $\Gamma \vdash \mathrm{app}^*_{(x:A)B}(M,N) : C$ must end in an application of (Π^*-E) followed by a number of applications of the rules (Bu-Conv), (Ty-Conv), (Cong) and (Subst). We continue by induction on the number of application of these rules. If the last rule is (Π^*-E) then the assertion is immediate. For the induction step there are four possible cases for the last rule:

- (Bu-Conv), say with equality $\vdash \Gamma = \Gamma'$ Bunch. By induction hypothesis, we have $\Delta'_\circ$, Φ' and Ψ' with $\Gamma' \equiv \Delta'(\Phi' * \Psi')$, $\Phi' * x\colon A \vdash B$ Type, $\Gamma' \vdash C = \Pi^* x\colon A.B$ Type, $\Phi' \vdash M : \Pi^* x\colon A.B$ and $\Psi' \vdash N : A$. By Lemma 4.4.8 there exists a context Θ with $\vdash \Theta = \Delta'(\Phi' * \Psi')$ Bunch and $\Theta \equiv \Gamma$. Since bunch equality does not manipulate the bunching structure, Θ must have the form $\Delta(\Phi * \Psi)$ and $\vdash \Phi = \Phi'$ Bunch and $\vdash \Psi = \Psi'$ Bunch. The assertion then follows, since using bunch conversion we can derive $\Phi * x\colon A \vdash B$ Type, $\Gamma \vdash C = \Pi^* x\colon A.B$ Type, $\Phi \vdash M : \Pi^* x\colon A.B$ and $\Psi \vdash N : A$.
- (Ty-Conv). Immediate from the induction hypothesis and transitivity of type equality.
- (Cong), say with $\Gamma \succcurlyeq \Gamma'$. By induction hypothesis, we have $\Delta'_\circ$, Φ' and Ψ' with $\Gamma' \equiv \Delta'(\Phi' * \Psi')$, $\Phi' * x\colon A \vdash B$ Type, $\Gamma' \vdash C = \Pi^* x\colon A.B$ Type, $\Phi' \vdash M : \Pi^* x\colon A.B$ and $\Psi' \vdash N : A$. By transitivity of $\succcurlyeq$ we get $\Gamma \succcurlyeq \Delta'(\Phi' * \Psi')$. By Lemma 4.4.6, there exist Δ, Φ and Ψ with $\vdash \Delta(\Phi * \Psi)$ Bunch, $\Phi \succcurlyeq \Phi'$, $\Psi \succcurlyeq \Psi'$ and $\Gamma \equiv \Delta(\Phi * \Psi)$. Using (Cong) we can derive the required $\Phi * x\colon A \vdash B$ Type, $\Gamma \vdash C = \Pi^* x\colon A.B$ Type, $\Phi \vdash M : \Pi^* x\colon A.B$ and $\Psi \vdash N : A$.

- (SUBST), say where Γ is $\Gamma'(\Gamma'', u\colon D)$ and the substituted term is $\Gamma'' \vdash R : D$. By induction hypothesis, we have $\Delta'_\circ$, Φ' and Ψ' with $\Gamma'(\Gamma'', u\colon D) \equiv \Delta'(\Phi' * \Psi')$ as well as $\Phi' * x\colon A \vdash B$ Type, $\Gamma' \vdash C = \Pi^* x\colon A.\, B$ Type, $\Phi' \vdash M : \Pi^* x\colon A.\, B$ and $\Psi' \vdash N : A$. By Lemma 4.4.7, there exist $\Delta_\circ$ and $\Gamma''' \succcurlyeq \Gamma''$ such that $\Delta'(\Phi' * \Psi')$ is $\Delta(\Gamma''', u\colon D)$. By (CONG), we have $\Gamma''' \vdash R : D$. The assertion follows by case distinction on whether u occurs in $\Delta'_\circ$, Φ' or Ψ', and in each case substituting appropriately.

□

At several points in later chapters, arguments can be simplified or generalised if the type theory has the strengthening property. Although we will not rely on strengthening, we give a definition of it here, so that we can state its consequences precisely.

Definition 4.4.10. A type theory has *strengthening* if, for all derivable judgements $\Gamma \vdash \mathscr{J}$ Type and $\Gamma \succcurlyeq \Phi$ with $FV(\mathscr{J}) \subseteq v(\Phi)$, it holds that $\Phi \vdash \mathscr{J}$ Type is derivable.

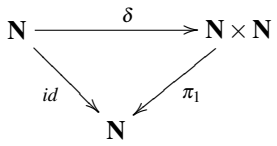
Not all type theories have this property. It can be violated by an appropriate choice of constants and axioms. For example, with constants $\vdash A$ Type, $\vdash B$ Type, $\vdash f : A$ and axiom $x\colon B \vdash A = B$ Type, we can derive $x\colon B \vdash f : B$ by weakening and type-conversion, but $\vdash f : B$ is not derivable.

4.5 Related and Further Work

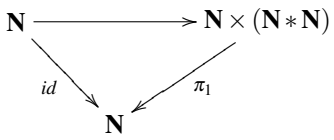
There are other possibilities for defining bunched dependent type theories. The motivation for the definition of bunches in this chapter was to find the simplest kind of bunches that allows us to work with Π and Π^* and that allows us to substitute for all variables. It is not strictly necessary to allow substitution for all variables. Cheney [20], for example, gives a sequent calculus for nominal logic over a simple type theory which uses only context multiplication of the form $\Gamma * x\colon A$. It may be interesting to investigate how useful a type theory with such restricted bunches would be.

Rather than restricting bunches even further, one may also ask for more expressive bunches. For example, one may ask for the context multiplication $*$ to be dependent to some degree. In [91, §15.15], Pym outlines a bunched type theory in which $*$ is dependent. The reason we propose a different system here is that in Pym's calculus the context operation $*$ has to be fibred. Although assuming $*$ to be fibred is natural from the type theoretic perspective, it excludes the models that we are interested in. For $*$ to be a fibred monoidal structure, it would have to be a monoidal structure on all the slices categories $\mathbb{B}/\Gamma$, not just on $\mathbb{B}$. In addition, it would have to be preserved under reindexing, i.e. $\sigma^*(A * B) \cong (\sigma^* A) * (\sigma^* B)$. In the intended semantics in the Schanuel topos, there is no way to extend the monoidal structure for freshness to a fibred monoidal structure. Suppose we had a fibred monoidal structure $*$ for the codomain fibration on $\mathbb{S}$, such that on $\mathbb{S}/1 \cong \mathbb{S}$ it coincides (up to isomorphism) with the monoidal structure for freshness from Section 3.3.1. To see that no such fibred monoidal structure $*$ can exist, consider the

slice category $\mathbb{S}/\mathbf{N}$ and the following morphism in it.



This morphism has the type $\delta\colon\ !^*1 \to !^*\mathbf{N}$ in $\mathbb{S}/\mathbf{N}$, where $!\colon \mathbf{N} \to 1$ and where we consider the object $\mathbf{N}$ of $\mathbb{S}$ as an object of $\mathbb{S}/1$. Using the monoidal structure $*$ on $\mathbb{S}/\mathbf{N}$, we get a morphism $\delta * \delta\colon\ (!^*1) * (!^*1) \to (!^*\mathbf{N}) * (!^*\mathbf{N})$ in $\mathbb{S}/\mathbf{N}$. Since a fibred monoidal structure is preserved by reindexing, this amounts to a map $!^*(1*1) \to !^*(\mathbf{N}*\mathbf{N})$ in $\mathbb{S}/\mathbf{N}$. Note that the monoidal structure $*$ used in both the domain and codomain of this map lives in the slice $\mathbb{S}/1 \cong \mathbb{S}$. By assumption, in the slice $\mathbb{S}/1$, the monoidal structure $*$ is just the monoidal structure for freshness defined in Chapter 3. As a consequence, we get $1*1 \cong 1$. Hence, the map $!^*(1*1) \to !^*(\mathbf{N}*\mathbf{N})$ amounts to a map $!^*1 \to !^*(\mathbf{N}*\mathbf{N})$ in $\mathbb{S}/\mathbf{N}$. Spelled out, this gives a commuting triangle of the following form in $\mathbb{S}$.



Note that this triangle would give us a map of type $\mathbf{N} \to \mathbf{N}*\mathbf{N}$ in $\mathbb{S}$. It is an easy consequence of Proposition 3.3.5 that there exists no map of this type in $\mathbb{S}$. Hence, the monoidal structure for freshness in the Schanuel topos cannot be extended to a fibred monoidal structure.

This semantic argument shows that calculi relying on the monoidal structure being fibred, such as the one of Pym [91, §15.15], cannot be used to work with the monoidal structure for freshness in $\mathbb{S}$. The problem is not that for given objects A and B in $\mathbb{S}/\Gamma$ there are no sensible choices of $A*B$ in $\mathbb{S}/\Gamma$. Rather, the problem is that no such choice can be functorial. Since functoriality is essential for substitution in bunches, it is likely that some approach other than bunches is needed to make this work.

Ishtiaq & Pym [92] define a dependent type theory containing both a linear and an additive function space. It is not clear what the precise relation of their type theory to the type theory presented here is. For example, to handle linearity Ishtiaq and Pym allow variables to be defined more than once in a context and need a binding strategy to determine by which variable in the context a free variable is bound. Besides the syntactic differences, the calculus in [92] has context extension rules

$$\frac{\Gamma \vdash A\ \text{Type}}{\Gamma, x\colon A\ \text{Context}} \qquad \frac{\Gamma \vdash A\ \text{Type}}{\Gamma, x!A\ \text{Context}}$$

in which $x!A$ corresponds to a linear variable. The dependencies in these rules suggest that linearity is interpreted in a fibred way, and we have argued above that there is a problem with interpreting this in the Schanuel topos.

Another linear dependent type theory is the linear logical framework of Cervesato and Pfenning [19]. This type theory extends the LF logical framework with a linear function space $A \multimap B$ that is both more

and less dependent than Π^*. It is less dependent because in $A \multimap B$ the type B is not allowed to depend on A, but it is more dependent because in $A \multimap B$ the type A need not be closed. Again, the dependencies in the context formation rules suggest that linearity is interpreted in a fibred way.

The dependent type theory in this section may be considered a generalisation of the $\alpha\lambda$-calculus of O'Hearn and Pym. The $\alpha\lambda$-calculus is a type theory for a category that is both cartesian closed and monoidal closed. Since $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$ is a type theory for a category that is locally cartesian closed and monoidal closed, it may be considered as extending the $\alpha\lambda$-calculus in the additive direction. The $\alpha\lambda$-calculus relates to $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$ in the same way the simply-typed λ-calculus relates to first-order dependent type theory. In this relationship, there is a mismatch between the way contexts can be formed, because, with dependent types, the additive context extension $(\Gamma, x\colon A)$ is asymmetric, while in the $\alpha\lambda$-calculus whole contexts can be joined additively (Γ,Δ). For the relation of the simply-typed λ-calculus to standard dependent type theory, this mismatch can be ignored: The simply-typed context $(x\colon A, (y\colon B, z\colon C))$, which is not available in dependent type theory, can simply be transformed to the context $((x\colon A, y\colon B), z\colon C)$, which is available. With bunches we have to be more careful. In $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$ we do not directly have the context $\Gamma * (y\colon B, z\colon C)$ of the $\alpha\lambda$-calculus. As discussed earlier, this context can nevertheless be encoded as $\Gamma * (z\colon B \times C)$. With $*$-types, that we will introduce in the next chapter, we can use this idea to encode the $\alpha\lambda$-calculus in $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$. We can also make $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ a direct super-system of the $\alpha\lambda$-calculus by adding the following rules for cartesian products. Although not formally checked, we believe that these rules are sound and form a conservative extension of $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$.

$$(\textsc{Bu-Cart})\ \frac{\vdash \Gamma\ \mathrm{Bunch} \qquad \vdash \Delta\ \mathrm{Bunch}}{\vdash \Gamma \times \Delta\ \mathrm{Bunch}}$$

$$(\textsc{Unit-Cart})\ \frac{\Gamma(\Delta) \vdash \mathscr{J}}{\Gamma(\Delta \times \diamond) \vdash \mathscr{J}} \qquad (\textsc{Swap-Cart})\ \frac{\Gamma(\Delta \times \Phi) \vdash \mathscr{J}}{\Gamma(\Phi \times \Delta) \vdash \mathscr{J}}$$

$$(\textsc{Assoc-Cart})\ \frac{\Gamma((\Delta \times \Phi) \times \Psi) \vdash \mathscr{J}}{\Gamma(\Delta \times (\Phi \times \Psi)) \vdash \mathscr{J}} \qquad (\textsc{Diag})\ \frac{\Gamma(\Delta \times \Delta') \vdash \mathscr{J}}{\Gamma(\Delta) \vdash \mathscr{J}[\nu(\Delta')/\nu(\Delta)]}$$

With this extension, the rules of (a version with type-labelled application of) the $\alpha\lambda$-calculus become derivable. Abstraction and application, for instance, can be derived as follows.

$$\begin{array}{r} (\textsc{Unit}), (\textsc{Unit-Cart}), (\textsc{Weak}) \\ (\textsc{Diag}) \\ (\Pi\text{-I}) \end{array} \frac{\begin{array}{c} \Gamma \times (x\colon A) \vdash M : B \\ \hline (\Gamma, x\colon A) \times (\Gamma', y\colon A) \vdash M[y/x] : B \end{array}}{\Gamma, x\colon A \vdash M : B} \over \Gamma \vdash \lambda x\colon A.M : A \to B$$

$$\begin{array}{r} (\textsc{Unit}), (\textsc{Weak}) \\ (\Pi\text{-E}) \end{array} \frac{\begin{array}{c} \Gamma \vdash M : A \to B \\ \hline \Gamma \times \Delta \vdash M : A \to B \end{array} \qquad \begin{array}{c} \Delta \vdash N : A \\ \hline \Gamma \times \Delta \vdash N : A \end{array}}{\Gamma \times \Delta \vdash \mathrm{app}_{(x:A)B}(M,N) : B}$$

For further work, we believe that it should not be hard to show decidability of type-checking for $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$ by adapting well-known type-checking techniques for logical frameworks [43, 25, 3].

Chapter 5

Monoidal Pair Types

In this chapter we extend the type theory $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$ with $*$-types. We show that $*$-types can be used to internalise context multiplication $\Gamma * \Delta$ in the same way that Σ-types can be used to internalise additive context concatenation $(\Gamma, x\colon A)$. Thus, by adding $*$-types we remove some of the defects of $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$ that we have discussed in the last chapter. In this chapter we also introduce a representation of simultaneous substitution for the bunched type theory with $*$-types and give substitution lemmas for it. This representation of substitution is used to make precise the internalisation of contexts. It will also be used for showing completeness in Chapter 7.

5.1 The System $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$

5.1.1 Syntax

We extend the syntax of $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$ with the following types and terms.

Types	$A ::= \cdots \mid A*A$
Terms	$M ::= \cdots \mid \mathrm{pair}^*_{A,A}(M,M) \mid \text{let } M \text{ be } x{:}A*x{:}A \text{ in } M$

$$FV(A*B) = FV(A) \cup FV(B)$$
$$FV(\mathrm{pair}^*_{A,B}(M,N)) = FV(A) \cup FV(B) \cup FV(M) \cup FV(N)$$
$$FV(\text{let } M \text{ be } x{:}A*y{:}B \text{ in } N) = FV(A) \cup FV(B) \cup FV(M) \cup (FV(N) \setminus \{x,y\})$$

We frequently omit type annotations for notational convenience.

$$\mathit{strip}(\mathrm{pair}^*_{A,B}(M,N)) = M*N$$
$$\mathit{strip}(\text{let } M \text{ be } x{:}A*y{:}B \text{ in } N) = \text{let } M \text{ be } x*y \text{ in } N$$

We use the following notation for the derivable inclusion of $A*B$ in $A\times B$ and the resulting projections.

$$\iota(M) \stackrel{\text{def}}{=} \text{let } M \text{ be } x{:}A*y{:}B \text{ in } \langle x,y\rangle$$
$$\dot{\pi}_1(M) \stackrel{\text{def}}{=} \text{let } M \text{ be } x{:}A*y{:}B \text{ in } x$$
$$\dot{\pi}_2(M) \stackrel{\text{def}}{=} \text{let } M \text{ be } x{:}A*y{:}B \text{ in } y$$

5.1.2 Rules for Monoidal Products

Rules marked with † will be discussed in the next section.

5.1.2.1 Rules for $*$-types

Formation

$$(*\text{-TY})^\dagger \ \frac{\vdash A \text{ Type} \qquad \vdash B \text{ Type}}{\vdash A*B \text{ Type}}$$

Introduction

$$(*\text{-I}) \ \frac{\vdash A \text{ Type} \qquad \vdash B \text{ Type} \qquad \Gamma \vdash M : A \qquad \Delta \vdash N : B}{\Gamma * \Delta \vdash \text{pair}^*_{A,B}(M,N) : A*B}$$

Elimination

$$(*\text{-E})^\dagger \ \frac{\begin{array}{c}\Gamma(z\colon A*B) \vdash C \text{ Type} \\ \Delta \vdash M : A*B \\ \Gamma(x\colon A*y\colon B)[\text{pair}^*_{A,B}(x,y)/z] \vdash N : C\,[\text{pair}^*_{A,B}(x,y)/z]\end{array}}{\Gamma(\Delta)\,[M/z] \vdash \text{let } M \text{ be } x{:}A*y{:}B \text{ in } N : C\,[M/z]}$$

Congruences

$$(*\text{-TY-CGR}) \ \frac{\vdash A_1 = A_2 \text{ Type} \qquad \vdash B_1 = B_2 \text{ Type}}{\Gamma \vdash A_1*B_1 = A_2*B_2 \text{ Type}}$$

$$(*\text{-I-CGR}) \ \frac{\begin{array}{cc}\vdash A_1 = A_2 \text{ Type} & \Gamma \vdash M_1 = M_2 : A_1 \\ \vdash B_1 = B_2 \text{ Type} & \Gamma \vdash N_1 = N_2 : B_2\end{array}}{\Gamma \vdash \text{pair}^*_{A_1,B_1}(M_1,N_1) = \text{pair}^*_{A_2,B_2}(M_2,N_2) : A_1*B_1}$$

$$(*\text{-E-CGR}) \ \frac{\begin{array}{cc}\vdash A_1 = A_2 \text{ Type} & \\ \vdash B_1 = B_2 \text{ Type} & \Gamma(z\colon A_1*B_1) \vdash C \text{ Type} \\ \Delta \vdash M_1 = M_2 : A_1*B_1 & \Gamma(x\colon A_1*y\colon B_1)[\text{pair}^*_{A_1,B_1}(x,y)/z] \vdash N_1 = N_2 : C[\text{pair}^*_{A_1,B_1}(x,y)/z]\end{array}}{\Gamma(\Delta)[M_1/z] \vdash \text{let } M_1 \text{ be } x{:}A_1*y{:}B_1 \text{ in } N_1 = \text{let } M_2 \text{ be } x{:}A_2*y{:}B_2 \text{ in } N_2 : C[M_1/z]}$$

Equations

$$(*\text{-}\beta)\ \frac{\begin{array}{ccc}\Gamma \vdash M : A & \Delta \vdash N : B & \Phi(z\colon A*B) \vdash C \text{ Type} \\ & & \Phi(x\colon A * y\colon B)[\text{pair}^*_{A,B}(x,y)/z] \vdash R : C[\text{pair}^*_{A,B}(x,y)/z]\end{array}}{\Phi(\Gamma * \Delta)[\text{pair}^*_{A,B}(M,N)/z] \vdash (\text{let pair}^*_{A,B}(M,N) \text{ be } x{:}A{*}y{:}B \text{ in } R) = (R\,[M/x]\,[N/y]) : C[\text{pair}^*_{A,B}(M,N)/z]}$$

$$(*\text{-}\eta)\ \frac{\Delta \vdash M : A * B \qquad \Gamma(z\colon A * B) \vdash N : C}{\Gamma(\Delta)[M/z] \vdash N\,[M/z] = \text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } (N\,[\text{pair}^*_{A,B}(x,y)/z]) : C[M/z]}$$

$$(\text{INJECT})^\dagger\ \frac{\begin{array}{ll}\Gamma \vdash M : A*B & \Gamma \vdash \dot{\pi}_1(M) = \dot{\pi}_1(N) : A \\ \Gamma \vdash N : A*B & \Gamma \vdash \dot{\pi}_2(M) = \dot{\pi}_2(N) : B\end{array}}{\Gamma \vdash M = N : A*B}$$

5.2 Discussion

The rules for $*$-types are such that the type $A*B$ can be formed only for closed types A and B. Since $*$ corresponds to a monoidal structure, it would be reasonable to expect the following type formation rule.

$$(*\text{-TY-WRONG})\ \frac{\Gamma \vdash A \text{ Type} \qquad \Delta \vdash B \text{ Type}}{\Gamma * \Delta \vdash A*B \text{ Type}}$$

In the codomain fibration, this rule would take the objects $\pi_A\colon A \to \Gamma$ and $\pi_B\colon B \to \Delta$ to the object $\pi_A * \pi_B\colon A*B \to \Gamma*\Delta$. The reason for *not* allowing this more general rule is that substitution on the type $A*B$ does not behave well. Consider just the special case of a substitution inside the context Γ. Clearly, it should not make a difference if we first do this substitution in $\Gamma \vdash A$ Type and then apply rule ($*$-TY-WRONG) or if we first apply the rule ($*$-TY-WRONG) and then do the substitution. Semantically, this means that, for any map $\sigma\colon \Gamma' \to \Gamma$, the objects $(\sigma^*\pi_A) * \pi_B$ and $(\sigma * \Delta)^*(\pi_A * \pi_B)$ in the slice over $\Gamma' * \Delta$ should be isomorphic. In the Schanuel topos, however, this is not the case.

$$\begin{array}{ccc}\mathbf{N} & \longrightarrow & 1 \\ {\scriptstyle \sigma^*\pi_A}\downarrow & \lrcorner & \downarrow{\scriptstyle \pi_A} \\ \mathbf{N} & \xrightarrow[\sigma]{} & 1\end{array} \qquad \begin{array}{ccc}\mathbf{N} & \longrightarrow & \mathbf{N} \\ {\scriptstyle \pi_B}\downarrow & \lrcorner & \downarrow{\scriptstyle \pi_B} \\ 1 & \longrightarrow & 1\end{array} \qquad \begin{array}{ccc}\mathbf{N}*\mathbf{N} & \longrightarrow & 1*\mathbf{N} \\ {\scriptstyle (\sigma^*\pi_A)*\pi_B}\downarrow & & \downarrow{\scriptstyle \pi_A*\pi_B} \\ \mathbf{N}*1 & \xrightarrow[\sigma*1]{} & 1*1\end{array}$$

The objects $(\sigma^*\pi_A)*\pi_B$ and $(\sigma*1)^*(\pi_A*\pi_B)$ are not isomorphic, since the left two squares are pullbacks while the right square is not. In this example, the map σ is in fact just a weakening map.

This shows that substitution on open types of the form $(A*B)$ cannot be defined in the naive way. Given the semantics of ($*$-TY-WRONG), one could argue that the type $(A*B)$ should be annotated with the contexts in which A and B are formed, i.e. we should write $A_\Gamma * B_\Delta$. Then the above problem with weakening disappears, since permuting weakening and ($*$-TY-WRONG) then changes the judgement in the conclusion. However, such annotations would make the term calculus quite complicated, which is why we do not consider them further. Nevertheless, we will meet a special case of ($*$-TY-WRONG) in

the shape of freefrom types in Chapter 8. In this special case, Δ is a context declaring a single variable and B is the unit type.

It should be pointed out that in the subobject logic of the Schanuel topos the above mentioned problem with substitution does not exist. If, for monomorphisms φ and ψ, the two squares on the left are pullbacks then so is the one on the right.

$$\begin{array}{ccc} A' & \longrightarrow & A \\ {\scriptstyle \sigma^*\varphi}\downarrow & \lrcorner & \downarrow{\scriptstyle \varphi} \\ \Gamma' & \xrightarrow[\sigma]{} & \Gamma \end{array} \qquad \begin{array}{ccc} B' & \longrightarrow & B \\ {\scriptstyle \tau^*\psi}\downarrow & \lrcorner & \downarrow{\scriptstyle \psi} \\ \Delta' & \xrightarrow[\tau]{} & \Delta \end{array} \qquad \begin{array}{ccc} A'*B' & \longrightarrow & A*B \\ {\scriptstyle (\sigma^*\varphi)*(\tau^*\psi)}\downarrow & \lrcorner & \downarrow{\scriptstyle \varphi*\psi} \\ \Gamma'*\Delta' & \xrightarrow[\sigma*\tau]{} & \Gamma*\Delta \end{array}$$

The rule ($*$-E) for $*$-elimination also deserves comment. We choose to formulate ($*$-E) with parameter context as opposed to the following rule without parameter context.

$$(*\text{-E}') \frac{\begin{array}{c} \Gamma * z\colon A*B \vdash C \text{ Type} \\ \Delta \vdash M : A*B \qquad \Gamma * x\colon A * y\colon B \vdash N : C\,[\text{pair}^*_{A,B}(x,y)/z] \end{array}}{\Gamma * \Delta \vdash \text{let } M \text{ be } x{:}A*y{:}B \text{ in } N : C\,[M/z]}$$

The main reason for choosing ($*$-E) over ($*$-E$'$) is that ($*$-E) allows us to derive the display property that terms $\Gamma(\Delta) \vdash M : A$ are in one-to-one correspondence with terms $\Delta \vdash M' : A'$ for some type A', see Section 5.6 below. Because of this property, we can omit the parameter contexts in the rules for all other type formers. Rules with parameter contexts become admissible. As a consequence, we have to deal with the problems arising from parameter contexts only once, for $*$-types. This helps, for example, in simplifying the semantic interpretation, which is easier in the absence of parameter contexts.

One problem caused by parameter contexts is that the rule ($*$-E) cannot always be permuted with substitution. This is a problem because it means that showing substitution is admissible is not straightforward, although we show in Section 7.2.3 that substitution can nevertheless be eliminated up to commuting conversion. To see why substitution and $*$-elimination can not always be permuted, consider the following derivation.

$$(\text{SUBST}) \frac{(*\text{-E}) \dfrac{z\colon A*B \vdash M : A*B \qquad (x\colon A * y\colon B), u\colon C[x*y/z] \vdash N : D[x*y/z]}{z\colon A*B, u\colon C[M/z] \vdash \text{let } M \text{ be } x*y \text{ in } N : D[M/z]} \qquad z\colon A*B \vdash R : C[M/z]}{z\colon A*B \vdash (\text{let } M \text{ be } x*y \text{ in } N)[R/u] : D[M/z][R/u]}$$

It is not always possible to transform this derivation such that (SUBST) is applied before ($*$-E). To apply the substitution before the $*$-elimination, we would have to find a term $x\colon A * y\colon B \vdash R' : C[x*y/z]$ (satisfying some conditions) which to substitute for u. The existence of a term $z\colon A*B \vdash R : C[M/z]$ does not imply the existence of an appropriate term R'. For example, let both A and B be the type $(1+1)$ in the Schanuel topos. Let the type $C(z)$ be such that $C(\kappa_1(\text{unit}) * \kappa_1(\text{unit}))$ is the empty type and $C(\kappa_2(\text{unit}) * \kappa_2(\text{unit}))$ is the unit type. Then, there can be no term $x\colon A * y\colon B \vdash R' : C[x*y/z]$, since substituting $\kappa_1(\text{unit})$ for x and y would give an element of the empty type. Note that we can take M to be $\kappa_2(\text{unit}) * \kappa_2(\text{unit})$, in which case the term R is just the unique element of the unit type. This

shows that the rules (SUBST) and ($*$-E) in the above derivation cannot be permuted. The rule ($*$-E′) without parameter contexts, on the other hand, does not have this problem. Substitution is easily seen to commute with it.

The problem that substitution does not commute with $*$-elimination is not unique to dependent types. In the simply typed $\alpha\lambda$-calculus of O'Hearn and Pym it also exists in the form that $*$-elimination does not commute with contraction. For example, the term $(\text{let } z \text{ be } x*y \text{ in } \langle x,z\rangle)$ is derivable by:

$$\dfrac{\dfrac{(x\colon A * y\colon B)\times(u\colon A*B)\vdash \langle x,u\rangle : A\times(A*B)}{(z\colon A*B)\times(u\colon A*B)\vdash \text{let } z \text{ be } x*y \text{ in } \langle x,u\rangle : A\times(A*B)}\,(*\text{-E})}{z\colon A*B\vdash \text{let } z \text{ be } x*y \text{ in } \langle x,z\rangle : A\times(A*B)}\,(\text{CONTR})$$

In this derivation, contraction and ($*$-E) cannot be exchanged. Without type dependency, the problem is nevertheless simpler than the problem with substitution above, since contraction is the only substitution that does not commute with ($*$-E). In other words, the problem appears also for simple types, but it can be simplified by adding a contraction rule. With type dependency we can still add a contraction rule, but this simplifies the problem only if we also restrict the rule ($*$-E) so that the type C cannot depend on z. In general, adding a contraction rule does not simplify the problem.

Since the let-terms introduced in ($*$-E) are a special kind of explicit substitution, and substitutions can, in general, not be permuted, it should not be unexpected that not all substitutions can be pushed through the rule ($*$-E). With normal substitutions, however, the fact that they cannot in general be permuted is not a problem, since for any given term we can always just perform them in order. This leads one to expect that let-terms can be moved out of the way of the substitutions. Take, for example, the above term $(\text{let } z \text{ be } x*y \text{ in } \langle x,z\rangle)$. It is not derivable without doing a substitution after a $*$-elimination. Using commuting conversions derived from the equations for $*$-types, we can prove this term to be equal to $\langle \text{let } z \text{ be } x*y \text{ in } x, \text{let } z \text{ be } x*y \text{ in } z\rangle$ and further to $\langle \text{let } z \text{ be } x*y \text{ in } x,z\rangle$. The last term is derivable without using substitution after $*$-elimination, so that by commuting conversion we have moved the let out of the way of the substitution. In Section 7.2.3 we will generalise this informal argument and show that, *up to commuting conversion*, we can remove substitution after ($*$-E).

In Section 7.2.3 we show that it is possible to give a formulation of $*$-types with the elimination rule ($*$-E′) instead of ($*$-E) that is essentially equivalent to the formulation in this chapter.

Leaving aside the technical details of the rule ($*$-E), it is reasonable to ask why we use let-terms at all. Since we assume the monoidal structure $*$ to be affine, we have projections $\dot{\pi}_1 : A*B \to A$ and $\dot{\pi}_2 : A*B \to B$ which could be added to the syntax directly.

$$\dfrac{\Gamma\vdash M : A*B}{\Gamma\vdash \dot{\pi}_1(M) : A} \qquad \dfrac{\Gamma\vdash M : A*B}{\Gamma\vdash \dot{\pi}_2(M) : B}$$

This formulation is weaker than that with let-terms. For example, the above two elimination rules are not sufficient to derive a term corresponding to the functoriality of $*$. With let-terms, the functoriality of $*$ is derivable as follows.

$$(*\text{-E})\ \frac{z\colon A*B \vdash z : A*B \qquad (*\text{-I})\ \dfrac{\vdash C, D\ \text{Type} \qquad x\colon A \vdash M : C \qquad y\colon B \vdash N : D}{x\colon A * y\colon B \vdash M*N : C*D}}{z\colon A*B \vdash \text{let } z \text{ be } x*y \text{ in } M*N : C*D}$$

The corresponding term with affine projections would be

$$z\colon A*B \vdash (M[\dot{\pi}_1(z)/x])*(N[\dot{\pi}_2(z)/y]) : C*D,$$

but this term is not derivable. This problem could be fixed by using the following alternative elimination rule for the affine projections. This rule is just a version of $(*\text{-E})$.

$$\frac{\Gamma(z\colon A*B) \vdash C\ \text{Type} \qquad \Delta \vdash M : A*B \qquad \Gamma(x\colon A * y\colon B)[x*y/z] \vdash N : C\,[x*y/z]}{\Gamma(\Delta)\,[M/z] \vdash N[\dot{\pi}_1(M)/x][\dot{\pi}_2(M)/y] : C\,[M/z]}$$

However, this rule is not very natural for the affine projections. First, it is not syntax-directed. The term in its conclusion may have any term constructor as its outermost connective. For example, there is a derivation of

$$z\colon A*B \vdash (M[\dot{\pi}_1(z)/x])*(N[\dot{\pi}_2(z)/y]) : C*D$$

having the above elimination as last rule. Furthermore, even though the term $(M[\dot{\pi}_1(z)/x])*(N[\dot{\pi}_2(z)/y])$ has $*$ as its outermost constructor, there is no derivation of it using $(*\text{-I})$ as the last logical rule. This is the case since $(*\text{-I})$ can only derive terms $M*N$ in which the free variables of M and N occur in separated parts of the context. In contrast, the formulation with let-terms is syntax-directed and all terms of the form $M*N$ are derived with $(*\text{-I})$ as the last logical rule; see the inversion lemma below.

In the formulation with let-terms it is also easier to understand *why* a given term is typeable. For example, with let-terms we can type a term of the form $M*N$ only if the free variables in M and N occur in separated parts of the context. The context can be manipulated using let. Therefore, it can be read off the terms why separation assertions hold. The price to pay for this is that let-terms make the terms larger. In the formulation with projections, on the other hand, it is harder to understand how a term has been typed. In the term $(M[\dot{\pi}_1(z)/x])*(N[\dot{\pi}_2(z)/y])$, for example, the projections $\dot{\pi}_1(z)$ and $\dot{\pi}_2(z)$ may occur anywhere and repeatedly in the terms M and N. The term $(M[\dot{\pi}_1(z)/x])*(N[\dot{\pi}_2(z)/y])$ itself does not tell us how $M[\dot{\pi}_1(z)/x]$ and $N[\dot{\pi}_2(z)/y]$ were separated—we have to guess the separation. Another good example is the term derived in Section 11.3.3.3, for which without let-terms it would not be clear at all how to derive it.

A final reason for choosing let-terms is that at present we do not know how to state the equations for the version with projections. It may well be the case that we can just take the usual β and η-equations, but this remains to be worked out.

Having given reasons for preferring let-terms, we point out that the formulations with projections also has its uses and should not be discarded altogether. It will be a useful tool for proving soundness in the next chapter. The reason why it is useful there is that the translation $(\text{let } M \text{ be } x*y \text{ in } N) \mapsto N[\dot{\pi}_1(M)/x][\dot{\pi}_2(M)/y]$ maps terms that differ only up to commuting conversion to equal terms. In the next chapter we will use this translation and show soundness both for the system with let-terms and

the system with projections. Because the translation identifies terms that differ only up to commuting conversion, there may be other uses of the formulation with projections. For example, we hope that it may help in giving an algorithm for deciding equality that does not have to deal with commuting conversions, see the further work section of this chapter. Finally, it should also be pointed out that a let-free formulation has been used in the type system of FreshML 2000 [87].

Regarding the equations for $*$-types, we highlight the rule (INJECT). This rule, whose premises are typeable only with (UNIT), amounts to the information that the canonical maps $A * B \to A \times B$ are monomorphic. Hence, the rules (UNIT) and (INJECT) make the monoidal structure $*$ strict affine.

5.3 Basic Properties

The basic properties, Lemmas 4.4.1–4.4.5, continue to hold for $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$.

Lemma 5.3.1 (Term Inversion)**.** *In addition to the cases from Lemma 4.4.9 the following are true.*

1. *If* $\Gamma \vdash \mathrm{pair}^*_{A,B}(M,N) : C$ *then there exist* $\Delta_\circ$, Φ, Ψ *such that* $\Gamma \equiv \Delta(\Phi * \Psi)$ *and* $\vdash A$ Type *and* $\vdash B$ Type *and* $\Gamma \vdash C = A*B$ Type *and* $\Phi \vdash M : A$ *and* $\Psi \vdash N : B$.

2. *If* $\Gamma \vdash$ let M be $x{:}A{*}y{:}B$ in $N : C$ *then there exist* $\Delta_\circ$, Φ, N' *and* D *such that the judgements* $\Delta'(z\colon A{*}B) \vdash D$ Type, $\Phi \vdash M : A{*}B$ *and* $\Delta'(x\colon A * y\colon B)\,[\mathrm{pair}^*_{A,B}(x,y)/z] \vdash N' : D\,[\mathrm{pair}^*_{A,B}(x,y)/z]$ *are derivable, and* $\Gamma \vdash$ let M be $x{:}A{*}y{:}B$ in $N : C$ *can be derived from these sequents by first using* ($*$-E)*, followed by a number of uses of type conversion, substitution and the structural rules.*

Proof. The case for $\mathrm{pair}^*_{A,B}(M,N)$ is like that for $\mathrm{app}^*_{(x:A)B}(M,N)$ in Lemma 4.4.9. The case for let follows because bunch conversion commutes with all other rules. □

5.4 Eliminating $*$-types over Types

In this section we define a let on types, i.e. we define (let M be $x{:}A{*}y{:}B$ in C) where C is a type. We show that rules analogous to ($*$-E), ($*$-β) and ($*$-η) are admissible for this let on types.

We define (let M be $x{:}A{*}y{:}B$ in C) by induction on the type C.

$$\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } 1 \stackrel{\text{def}}{=} 1$$

$$\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } T(N) \stackrel{\text{def}}{=} T(\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } N)$$

$$\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } C{*}D \stackrel{\text{def}}{=} C{*}D$$

$$\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } \Pi z\colon C.\,D \stackrel{\text{def}}{=} \Pi z\colon (\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } C).\,(\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } D)$$

$$\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } \Sigma z\colon C.\,D \stackrel{\text{def}}{=} \Sigma z\colon (\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } C).\,(\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } D)$$

$$\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } \Pi^* z\colon C.\,D \stackrel{\text{def}}{=} \Pi^* z\colon C.\,(\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } D)$$

Having defined let on types, it is natural to extend the definition to contexts as well. Using let on types, any context $\Gamma(x\colon A*y\colon B)$ can be transformed into a context $\Gamma'(z\colon A*B)$ by replacing each type C in $\Gamma_\circ$ that may depend on x and y by the type (let z be $x{:}A*y{:}B$ in C). More precisely, the context-with-hole $\Gamma'_\circ$ is obtained from $\Gamma_\circ$ by means of the following extension of let to contexts-with-hole.

$$\begin{aligned}
\text{let } M \text{ be } x{:}A*y{:}B \text{ in } \bigcirc &= \bigcirc \\
\text{let } M \text{ be } x{:}A*y{:}B \text{ in } (\Gamma_\circ * \Gamma') &= (\text{let } M \text{ be } x{:}A*y{:}B \text{ in } \Gamma_\circ) * \Gamma' \\
\text{let } M \text{ be } x{:}A*y{:}B \text{ in } (\Gamma * \Gamma'_\circ) &= \Gamma * (\text{let } M \text{ be } x{:}A*y{:}B \text{ in } \Gamma'_\circ) \\
\text{let } M \text{ be } x{:}A*y{:}B \text{ in } (\Gamma_\circ, u\colon C) &= (\text{let } M \text{ be } x{:}A*y{:}B \text{ in } \Gamma_\circ), u\colon (\text{let } M \text{ be } x{:}A*y{:}B \text{ in } C)
\end{aligned}$$

In the rest of this section we show that the definition of let on types makes type versions of the rules ($*$-E), ($*$-E-CGR), ($*$-β) and ($*$-η) admissible. The rules are summarised at the end of this section. We give a series of lemmas leading up to these rules.

The first lemma contributes to the rules ($*$-E-TY) and ($*$-β-TY) at the end of this section.

Lemma 5.4.1. *If* $\vdash \Gamma(z\colon A*B)$ Bunch *and* $\Gamma(x\colon A*y\colon B)[\mathrm{pair}^*_{A,B}(x,y)/z] \vdash C$ Type *then the following judgements are derivable.*

$$\Gamma(z\colon A*B) \vdash \text{let } z \text{ be } x{:}A*y{:}B \text{ in } C \text{ Type}$$
$$\Gamma(x\colon A*y\colon B)[\mathrm{pair}^*_{A,B}(x,y)/z] \vdash C = (\text{let } \mathrm{pair}^*_{A,B}(x,y) \text{ be } x{:}A*y{:}B \text{ in } C) \text{ Type}$$

Proof. The proof goes by induction on the type C.

- C is 1. This case is trivial, since (let M be $x{:}A*y{:}B$ in 1) is defined to be 1.
- C is $C*D$. This case is immediate since (let M be $x{:}A*y{:}B$ in $C*D$) is defined to be $C*D$.
- C is $T(M)$. By Lemma 4.4.4, we have $\Gamma(x\colon A*y\colon B)[\mathrm{pair}^*_{A,B}(x,y)/z] \vdash M : D$ and $\vdash u\colon D$ Bunch and $T \in \mathscr{T}(u\colon D)$. Then $FV(D) = \emptyset$ holds, so that D is $D[\mathrm{pair}^*_{A,B}(x,y)/z]$. We can therefore use ($*$-E) to derive $\Gamma(z\colon A*B) \vdash$ let z be $x{:}A*y{:}B$ in $M : D$. Using (C-TY), this gives $\Gamma(z\colon A*B) \vdash T(\text{let } z \text{ be } x{:}A*y{:}B \text{ in } M)$ Type. But this type is the same as (let z be $x{:}A*y{:}B$ in $T(M)$), so that we have the first of the two required judgements. Since we have $\Gamma(x\colon A*y\colon B)[\mathrm{pair}^*_{A,B}(x,y)/z] \vdash M = \text{let } \mathrm{pair}^*_{A,B}(x,y) \text{ be } x{:}A*y{:}B \text{ in } M : D$ by ($*$-β), the second judgement follows using the congruence rule (C-TY-CGR).
- C is $\Pi u\colon C.D$. By Lemma 4.4.4, we have derivations of $\Gamma(x\colon A*y\colon B)[\mathrm{pair}^*_{A,B}(x,y)/z] \vdash C$ Type and $\Gamma(x\colon A*y\colon B)[\mathrm{pair}^*_{A,B}(x,y)/z], u\colon C \vdash D$ Type. From the induction hypothesis we get

  $$\Gamma(z\colon A*B) \vdash \text{let } z \text{ be } x{:}A*y{:}B \text{ in } C \text{ Type}$$
  $$\Gamma(x\colon A*y\colon B)[\mathrm{pair}^*_{A,B}(x,y)/z] \vdash C = \text{let } \mathrm{pair}^*_{A,B}(x,y) \text{ be } x{:}A*y{:}B \text{ in } C \text{ Type}$$

  We write Γ' for $\Gamma(x\colon A*y\colon B)[\mathrm{pair}^*_{A,B}(x,y)/z]$ and C' for (let z be $x{:}A*y{:}B$ in C). Using the rule (BU-AEQ) we get $\vdash (\Gamma', u\colon C) = (\Gamma', u\colon C'[\mathrm{pair}^*_{A,B}(x,y)/z])$ Bunch. By bunch conversion we

obtain $\Gamma', u\colon C'[\mathrm{pair}^*_{A,B}(x,y)/z] \vdash D$ Type, to which we can apply the induction hypothesis. It gives

$$\Gamma(z\colon A{*}B), u\colon C' \vdash \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } D \text{ Type}$$
$$\Gamma', u\colon C'[\mathrm{pair}^*_{A,B}(x,y)/z] \vdash D = \text{let } \mathrm{pair}^*_{A,B}(x,y) \text{ be } x{:}A{*}y{:}B \text{ in } D \text{ Type}.$$

We write D' for (let z be $x{:}A{*}y{:}B$ in D). Rule (Π-TY) gives $\Gamma(z\colon A{*}B) \vdash \Pi u\colon C'.D'$ Type, which by the definition of let on Π-types is just the first required judgement. The second judgement follows, since (Π-TY-CGR) allows us to derive $\Gamma' \vdash \Pi u\colon C.D = (\Pi u\colon C'.D')[\mathrm{pair}^*_{A,B}(x,y)/z]$ Type.

- C is $\Sigma u\colon C.D$. Similar to the previous case.

- C is $\Pi^* u\colon C.D$. Lemma 4.4.4 gives $\vdash C$ Type and $\Gamma(x\colon A * y\colon B)[\mathrm{pair}^*_{A,B}(x,y)/z] * u\colon C \vdash D$ Type. We get $FV(C) = \emptyset$, so that $C[\mathrm{pair}^*_{A,B}(x,y)/z]$ is C. We can therefore apply the induction hypothesis to obtain the following sequents, in which we write Γ' for $\Gamma(x\colon A * y\colon B)[\mathrm{pair}^*_{A,B}(x,y)/z]$.

$$\Gamma(z\colon A{*}B) * u\colon C \vdash \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } D \text{ Type}$$
$$\Gamma' * u\colon C \vdash D = \text{let } \mathrm{pair}^*_{A,B}(x,y) \text{ be } x{:}A{*}y{:}B \text{ in } D \text{ Type}.$$

We write D' for (let z be $x{:}A{*}y{:}B$ in D). Rule (Π^*-TY) gives $\Gamma(z\colon A{*}B) \vdash \Pi^* u\colon C.D'$ Type, which by the definition of let on Π^*-types is just the first required judgement. For the second judgement, we use (Π^*-TY-CGR) to derive the required $\Gamma' \vdash \Pi^* u\colon C.D = (\Pi^* u\colon C.D')[\mathrm{pair}^*_{A,B}(x,y)/z]$ Type.

□

Next we state a lemma justifying the rule ($*$-η-TY) at the end of this section.

Lemma 5.4.2. *If* $\Gamma(z\colon A{*}B) \vdash C$ Type *then the following sequent is derivable.*

$$\Gamma(z\colon A{*}B) \vdash C = (\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } C[\mathrm{pair}^*_{A,B}(x,y)/z]) \text{ Type}$$

Proof. The proof goes by induction on C.

- The cases where C is 1 or $D{*}E$ are immediate, since in both these cases the type C must be closed and (let z be $x{:}A{*}y{:}B$ in C) is defined to be C.

- C is $T(M)$. By Lemma 4.4.4, we have $\Gamma(z\colon A{*}B) \vdash M : D$ and $\vdash u\colon D$ Bunch and $T \in \mathscr{T}(u\colon D)$. By ($*$-$\eta$) we have $\Gamma(z\colon A{*}B) \vdash M = \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } M[\mathrm{pair}^*_{A,B}(x,y)/z] : D$. Since, by definition, (let z be $x{:}A{*}y{:}B$ in $T(M)[\mathrm{pair}^*_{A,B}(x,y)/z]$) is just T(let z be $x{:}A{*}y{:}B$ in $M[\mathrm{pair}^*_{A,B}(x,y)/z]$), the assertion then follows using (C-TY-CGR).

- C is $\Pi u\colon C.D$. By Lemma 4.4.4, we have $\Gamma(z\colon A{*}B) \vdash C$ Type and $\Gamma(z\colon A{*}B), u\colon C \vdash D$ Type. From the induction hypothesis we get

$$\Gamma(z\colon A{*}B) \vdash C = \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } C[\mathrm{pair}^*_{A,B}(x,y)/z] \text{ Type}$$
$$\Gamma(z\colon A{*}B), u\colon C \vdash D = \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } D[\mathrm{pair}^*_{A,B}(x,y)/z] \text{ Type}.$$

Since $(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } (\Pi u\colon C.D)[\text{pair}^*_{A,B}(x,y)/z])$ is defined to be

$$(\Pi u\colon (\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } C[\text{pair}^*_{A,B}(x,y)/z]).(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } D[\text{pair}^*_{A,B}(x,y)/z])),$$

the assertion follows using (Π-TY-CGR).

- The cases where C is $\Sigma u\colon C.D$ or $\Pi^* u\colon C.D$ are similar to the previous case.

□

The previous two lemmas for let on types can be lifted to let on contexts.

Lemma 5.4.3. *If* $\vdash \Gamma(x\colon A*y\colon B)$ Bunch *then the following judgements are derivable.*

$$\vdash (\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } \Gamma_\circ)(z\colon A{*}B) \text{ Bunch}$$
$$\vdash (\text{let pair}^*_{A,B}(x,y) \text{ be } x{:}A{*}y{:}B \text{ in } \Gamma_\circ)(x\colon A*y\colon B) = \Gamma(x\colon A*y\colon B) \text{ Bunch}$$

Proof. Each of the judgements follows by induction on the context-with-hole $\Gamma_\circ$, using Lemma 5.4.1. □

Lemma 5.4.4. *If* $\vdash \Gamma(z\colon A{*}B)$ Bunch *then the following judgement is derivable.*

$$\vdash \Gamma(z\colon A{*}B) = (\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } \Gamma_\circ[\text{pair}^*_{A,B}(x,y)/z])(z\colon A{*}B) \text{ Bunch}$$

Proof. By induction on the context-with-hole $\Gamma_\circ$, using Lemma 5.4.2. □

To complete the rules for let on types, it just remains to show the congruence equation for let on types. For this we need the following substitution lemma.

Lemma 5.4.5. *If* $\Gamma(x\colon A*y\colon B) \vdash M : C$ *and* $u\colon C \vdash D$ Type *then*

$$(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } \Gamma_\circ)(z\colon A{*}B) \vdash \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } (D[M/u]) = D[\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } M/u] \text{ Type}.$$

Proof. In the proof we write Φ for $(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } \Delta(\Gamma_\circ))(z\colon A{*}B)$ and M_z for $(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } M)$. We show the stronger result that if the judgements $\Gamma(x\colon A*y\colon B) \vdash M : C$ and $\Delta(u\colon C) \vdash D$ Type are derivable then so is the sequent $\Phi \vdash \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } (D[M/u]) = D[M_z/u]$ Type. Note that $\Delta(u\colon C)$ is a context only if C is a closed type. The proof goes by induction on D.

- The cases where D is 1 or $E{*}F$ follow easily, since D is closed and $(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } D)$ is defined to be D.

- D is $T(N)$. By Lemma 4.4.4 we have $\Delta(u\colon C) \vdash N : E$ and $\vdash v\colon E$ Bunch and $T \in \mathscr{T}(v\colon E)$. Using Lemma 5.4.3 and the rules ($*$-η), ($*$-β), ($*$-E-CGR) and (SUBST-TM-CGR), we get

  $$\Phi \vdash N[M_z/u] = \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } N[M/u] : E.$$

  Since, by definition, $(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } T(N[M/u]))$ is just $T(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } N[M/u])$, the assertion then follows from this equation and (C-TY-CGR).

- D is $\Pi v\colon E.F$. By Lemma 4.4.4 we have $\Delta(u\colon C) \vdash E$ Type and $\Delta(u\colon C), v\colon E \vdash F$ Type. From the induction hypothesis we get

$$\Phi \vdash \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } E[M/u] = E[M_z/u] \text{ Type}$$
$$\Phi, v\colon (\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } E) \vdash \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } F[M/u] = F[M_z/u] \text{ Type}.$$

  Since $(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } (\Pi v\colon E.F))$ is defined to be

$$\Pi v\colon (\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } E).(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } F),$$

  the assertion follows using (Π-TY-CGR).

- The cases where D is $\Sigma u\colon E.F$ or $\Pi^* u\colon E.F$ are similar to the case for $\Pi u\colon E.F$.

□

Using this lemma, we can now show congruence for let on types.

Lemma 5.4.6. *If* $\vdash A_1 = A_2$ Type, $\vdash B_1 = B_2$ Type *and* $\Gamma(x\colon A_1 * y\colon B_1) \vdash C_1 = C_2$ Type *then*

$$(\text{let } z \text{ be } x{:}A_1{*}y{:}B_1 \text{ in } \Gamma_\circ)(z\colon A_1{*}B_1) \vdash \text{let } z \text{ be } x{:}A_1{*}y{:}B_1 \text{ in } C_1 = \text{let } z \text{ be } x{:}A_2{*}y{:}B_2 \text{ in } C_2 \text{ Type}.$$

Proof. We show that, for any derivation of some sequent $\Delta \vdash D_1 = D_2$ Type, from which we can derive the sequent $\Gamma(x\colon A_1 * y\colon B_1) \vdash C_1 = C_2$ Type using only the structural rules, substitution and bunch conversion, the judgement $(\text{let } z \text{ be } x{:}A_1{*}y{:}B_1 \text{ in } \Gamma_\circ)(z\colon A_1{*}B_1) \vdash \text{let } z \text{ be } x{:}A_1{*}y{:}B_1 \text{ in } C_1 = \text{let } z \text{ be } x{:}A_2{*}y{:}B_2 \text{ in } C_2$ Type is derivable. The assertion follows as a special case. In the rest of this proof we use Γ_z^i to abbreviate $(\text{let } z \text{ be } x{:}A_i{*}y{:}B_i \text{ in } \Gamma_\circ)(z\colon A_i{*}B_i)$, where $i \in \{1,2\}$. The proof goes by induction on the size of the derivation of $\Delta \vdash D_1 = D_2$ Type. We continue by case distinction on the last rule of that derivation.

- In the case where the derivation ends in a structural rule, substitution or bunch conversion, the induction hypothesis can be applied directly.

- (TY-REFL), (TY-SYM), (TY-TRANS) are straightforward.

- (TY-EQ-AX). The last rule is

$$\frac{\Delta \vdash M : A \qquad u\colon A \vdash B \text{ Type} \qquad u\colon A \vdash C \text{ Type}}{\Delta \vdash B[M/u] = C[M/u] \text{ Type}} \; (u\colon A \vdash B = C \text{ Type}) \in \mathscr{T}^{\mathscr{E}}(u\colon A)$$

  where D_1 is $B[M/u]$ and D_2 is $C[M/u]$. We apply to the left premise the structural rules, substitution and bunch conversion that lead from the conclusion to $C_1 = C_2$. From this we obtain a derivation of $\Gamma(x\colon A_1 * y\colon B_1) \vdash M' : A$ for some M' (note that A must be closed). Now note that C_1 is $B[M'/u]$ and C_2 is $C[M'/u]$. By (TY-EQ-AX), we have

$$\Gamma_z^1 \vdash B[\text{let } z \text{ be } x{:}A_1{*}y{:}B_1 \text{ in } M'/u] = C[\text{let } z \text{ be } x{:}A_1{*}y{:}B_1 \text{ in } M'/u] \text{ Type}$$

Using ($*$-E-CGR) and (SUBST-TY-CGR), this implies

$$\Gamma^1_z \vdash B[\text{let } z \text{ be } x{:}A_1 * y{:}B_1 \text{ in } M'/u] = C[\text{let } z \text{ be } x{:}A_2 * y{:}B_2 \text{ in } M'/u] \text{ Type}$$

By Lemma 5.4.5 we have

$$\Gamma^1_z \vdash \text{let } z \text{ be } x{:}A_1 * y{:}B_1 \text{ in } (B[M'/u]) = B[\text{let } z \text{ be } x{:}A_1 * y{:}B_1 \text{ in } M'/u] \text{ Type}$$
$$\Gamma^2_z \vdash \text{let } z \text{ be } x{:}A_2 * y{:}B_2 \text{ in } (C[M'/u]) = C[\text{let } z \text{ be } x{:}A_2 * y{:}B_2 \text{ in } M'/u] \text{ Type.}$$

It is a straightforward induction on $\Gamma_\circ$ to derive $\vdash \Gamma^1_z = \Gamma^2_z$ Bunch. Hence, transitivity gives the required equation

$$\Gamma^1_z \vdash \text{let } z \text{ be } x{:}A_1 * y{:}B_1 \text{ in } (B[M'/u]) = \text{let } z \text{ be } x{:}A_2 * y{:}B_2 \text{ in } (C[M'/u]) \text{ Type.}$$

- (Π-TY-CGR). The last rule is

$$(\Pi\text{-TY-CGR})\ \frac{\Delta \vdash E_1 = E_2 \text{ Type} \qquad \Delta, v\colon E_1 \vdash F_1 = F_2 \text{ Type}}{\Delta \vdash \Pi v\colon E_1.F_1 = \Pi v\colon E_2.F_2 \text{ Type}}$$

where D_1 is $\Pi v\colon E_1.F_1$ and D_2 is $\Pi v\colon E_2.F_2$. Using the induction hypothesis, we can derive

$$\Gamma^1_z \vdash \text{let } z \text{ be } x{:}A_1 * y{:}B_1 \text{ in } E'_1 = \text{let } z \text{ be } x{:}A_2 * y{:}B_2 \text{ in } E'_2 \text{ Type}$$
$$\Gamma^1_z, v\colon (\text{let } z \text{ be } x{:}A_1 * y{:}B_1 \text{ in } E'_1) \vdash \text{let } z \text{ be } x{:}A_1 * y{:}B_1 \text{ in } F'_1 = \text{let } z \text{ be } x{:}A_2 * y{:}B_2 \text{ in } F'_2 \text{ Type,}$$

where E'_i and F'_i arise from E_i and F_i by applying the substitutions that are applied after the rule (Π-TY-CGR) in the given derivation. Hence, C_1 is $\Pi v\colon E'_1.F'_1$ and C_2 is $\Pi v\colon E'_2.F'_2$. By using (Π-TY-CGR) and the definition of let on Π-types, we therefore obtain the required equation $\Gamma^1_z \vdash \text{let } z \text{ be } x{:}A_1 * y{:}B_1 \text{ in } C_1 = \text{let } z \text{ be } x{:}A_2 * y{:}B_2 \text{ in } C_2$ Type.

- The cases for the other congruence rules are similar to the case for (Π-TY-CGR).

□

By means of the above lemmas, we have shown admissible the following rules.

$$(*\text{-E-TY})\ \frac{\Delta \vdash M : A * B \qquad \Gamma(x\colon A * y\colon B) \vdash C \text{ Type}}{(\text{let } M \text{ be } x{:}A * y{:}B \text{ in } \Gamma_\circ)(\Delta) \vdash (\text{let } M \text{ be } x{:}A * y{:}B \text{ in } C) \text{ Type}}$$

$$(*\text{-E-TY-CGR})\ \frac{\begin{array}{cc} \vdash A_1 = A_2 \text{ Type} & \Delta \vdash M_1 = M_2 : A_1 * B_1 \\ \vdash B_1 = B_2 \text{ Type} & \Gamma(x\colon A_1 * y\colon B_1) \vdash C_1 = C_2 \text{ Type} \end{array}}{(\text{let } M_1 \text{ be } x{:}A_1 * y{:}B_1 \text{ in } \Gamma_\circ)(\Delta) \vdash \begin{array}{l} \text{let } M_1 \text{ be } x{:}A_1 * y{:}B_1 \text{ in } C_1 \\ = \text{let } M_2 \text{ be } x{:}A_2 * y{:}B_2 \text{ in } C_2 \end{array} \text{ Type}}$$

$$(*\text{-}\beta\text{-TY})\ \frac{\Gamma \vdash M : A \qquad \Delta \vdash N : B \qquad \Phi(x\colon A * y\colon B) \vdash C \text{ Type}}{\Phi(\Gamma * \Delta)[M/x][N/y] \vdash (\text{let pair}^*_{A,B}(M,N) \text{ be } x{:}A * y{:}B \text{ in } C) = (C[M/x][N/y]) \text{ Type}}$$

$$(*\text{-}\eta\text{-TY})\ \frac{\Delta \vdash M : A * B \qquad \Gamma(z\colon A * B) \vdash C\ \text{Type}}{\Gamma(\Delta)\,[M/z] \vdash C\,[M/z] = \text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } (C\,[\text{pair}^{*}_{A,B}(x,y)/z])\ \text{Type}}$$

$$(*\text{-E}^{+})\ \frac{\Delta \vdash M : A{*}B \qquad \Gamma(x\colon A * y\colon B) \vdash N : C}{(\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } \Gamma_{\circ})(\Delta) \vdash \text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } N : (\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } C)}$$

For example, to derive $(*\text{-E}^{+})$ from the above lemmas, we first use Lemma 5.4.3 and (BU-CONV) to obtain the sequent $(\text{let pair}^{*}_{A,B}(x,y) \text{ be } x{:}A{*}y{:}B \text{ in } \Gamma_{\circ})(x\colon A * y\colon B) \vdash N : C$. By Lemma 5.4.1 and (TY-CONV), we get $(\text{let pair}^{*}_{A,B}(x,y) \text{ be } x{:}A{*}y{:}B \text{ in } \Gamma_{\circ})(x\colon A * y\colon B) \vdash N : \text{let pair}^{*}_{A,B}(x,y) \text{ be } x{:}A{*}y{:}B \text{ in } C$. We can now apply the rule $(*\text{-E})$ to derive

$$(\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } \Gamma_{\circ})(z\colon A{*}B) \vdash \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } N : (\text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } C).$$

The conclusion of $(*\text{-E}^{+})$ then follows using (SUBST).

The importance of the above admissible rules is that, for any judgement $\Gamma(x\colon A * y\colon B) \vdash \mathscr{J}$ and any term $\Delta \vdash M : A{*}B$, we can derive the judgement $(\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } \Gamma_{\circ})(\Delta) \vdash \text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } \mathscr{J}$, no matter what the form of $\mathscr{J}$ is. We can therefore view 'let' as a special kind of substitution. The semantical counterpart of this view is that let is interpreted by substitution along the inverse of the map $[\text{pair}^{*}_{A,B}(x,y)/z]\colon (x\colon A) * (y\colon B) \to (z\colon A{*}B)$.

5.5 Representation of Substitution

In this section, we introduce a syntax to represent substitutions in the type theory. Such a syntax for substitutions is useful, for example to formulate the display property in the next section. It will also be used in the construction a term model for $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^{*})$ in Chapter 7.

The representation of substitution for the bunched type theory is more complicated than for normal dependent type theory, where simultaneous substitutions have a simple representation also known as telescopes [28]; see [110, 48]. One reason for the additional complexity is that not all the structural rules are admissible in **BT**, so that composition cannot be eliminated completely. Another reason why the representation of substitution for **BT** is more complicated is that we view 'let' as a substitution. This is because, in contrast to the other terms, 'let' cannot be described in terms of an ordinary substitution. Making 'let' a substitution is necessary for showing completeness of the syntax.

We define a representation of simultaneous substitution, the additive part of which is similar to the representations in [110, 109, 32].

$$\text{Substitutions} \qquad \sigma ::= \langle\rangle \mid \langle M/x\rangle \mid \langle M/x\colon A * x\colon A\rangle \mid \sigma \circ \sigma$$

The substitution $\langle M/x\rangle$ represents an ordinary substitution $(-)[M/x]$, while $\langle M/x\colon A * y\colon B\rangle$ represents

a let-term (let M be $x{:}A{*}y{:}B$ in $(-)$). The free variables are defined by

$$\begin{aligned} FV(\langle\rangle) &= \emptyset \\ FV(\langle M/x\rangle) &= FV(M) \cup \{x\} \\ FV(\langle M/x{:}\ A*y{:}\ B\rangle) &= FV(M) \cup \{x,y\} \\ FV(\sigma \circ \tau) &= FV(\sigma) \cup FV(\tau) \end{aligned}$$

We define the action of the representation of substitution on types and terms by

$$\begin{aligned} \mathscr{J}[\langle\rangle] &= \mathscr{J} \\ \mathscr{J}[\langle M/x\rangle] &= \mathscr{J}[M/x] \\ \mathscr{J}[\langle M/x{:}\ A*y{:}\ B\rangle] &= \text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } \mathscr{J} \\ \mathscr{J}[\sigma \circ \tau] &= \mathscr{J}[\sigma][\tau], \end{aligned}$$

where $\mathscr{J}$ may be a type A, a term M, or a context-with-hole $\Gamma_\circ$. Note that in this definition we make use of the let on types and on contexts-with-hole defined in the previous section. Note also that $\mathscr{J}[M/x]$ denotes ordinary syntactic substitution and not an explicit substitution.

Of course, not all substitutions σ are well-formed. There is no reason why, given a derivable judgement $\Gamma \vdash \mathscr{J}$, the judgement $\Delta \vdash \mathscr{J}[\sigma]$ should be derivable. It is not even clear what Δ should be. Therefore, we next the define a judgement $\sigma{:}\ \Delta \to \Gamma$ whose derivability ensures that $\Delta \vdash \mathscr{J}[\sigma]$ is derivable whenever $\Gamma \vdash \mathscr{J}$ is.

$$\text{(Sub-Id)}\ \frac{\vdash \Gamma \text{ Bunch}}{\langle\rangle : \Gamma \to \Gamma} \qquad \text{(Sub-Comp)}\ \frac{\sigma{:}\ \Gamma \to \Delta \qquad \tau{:}\ \Delta \to \Phi}{\tau \circ \sigma{:}\ \Gamma \to \Phi}$$

$$\text{(Sub-}*\text{)}\ \frac{\sigma{:}\ \Gamma \to \Delta \qquad \vdash \Phi \text{ Bunch}}{\sigma{:}\ (\Gamma * \Phi) \to (\Delta * \Phi)}\ FV(\Gamma,\sigma,\Delta) \cap FV(\Phi) = \emptyset$$

$$\text{(Sub-Lift)}\ \frac{\sigma{:}\ \Gamma \to \Delta \qquad \Delta \vdash A \text{ Type}}{\sigma{:}\ (\Gamma, x{:}\ A[\sigma]) \to (\Delta, x{:}\ A)}\ x \notin FV(\Gamma,\sigma,\Delta)$$

$$\text{(Sub-Sub)}\ \frac{\Gamma \vdash M : A}{\langle M/x\rangle : \Gamma \to (\Gamma, x{:}\ A)}\ x \notin FV(\Gamma)$$

$$\text{(Sub-Let)}\ \frac{\Gamma \vdash M : A{*}B}{\langle M/x{:}\ A*y{:}\ B\rangle : \Gamma \to (x{:}\ A)*(y{:}\ B)}\ x \neq y$$

$$\text{(Sub-Str)}\ \frac{\Gamma' \geqslant \Gamma \qquad \sigma{:}\ \Gamma \to \Delta \qquad \Delta \geqslant \Delta'}{\sigma{:}\ \Gamma' \to \Delta'}$$

$$\text{(Sub-Bu-Conv)}\ \frac{\vdash \Gamma' = \Gamma \text{ Bunch} \qquad \sigma{:}\ \Gamma \to \Delta \qquad \vdash \Delta = \Delta' \text{ Bunch}}{\sigma{:}\ \Gamma' \to \Delta'}$$

The side-conditions on the rules (SUB-$*$) and (SUB-LIFT) are important. Consider, for example, the substitution $\langle z/y\rangle \circ \langle x/z\rangle : x\colon A \to y\colon A$, which makes use of a temporary variable z. We do not want to allow this substitution to be lifted to a substitution of type $(x\colon A) * (z\colon B) \to (y\colon A) * (z\colon B)$, since any such lifted substitution should leave z untouched, and this is not the case for the given substitution.

Lemma 5.5.1. *The following rule is admissible.*

$$(\text{SUBST}^+)\ \frac{\sigma\colon \Gamma \to \Delta \qquad \Delta \vdash \mathscr{J}}{\Gamma \vdash \mathscr{J}[\sigma]}$$

Proof. By induction on the derivation of σ. □

It should be clear that (SUBST$^+$) subsumes the rules (SUBST), (WEAK), (UNIT), (SWAP), (ASSOC), ($*$-E$^+$) and (BU-CONV).

We define equality of substitutions.

Definition 5.5.2. Two substitutions $\sigma\colon \Gamma \to \Delta$ and $\tau\colon \Gamma \to \Delta$ are equal, written $\sigma = \tau\colon \Gamma \to \Delta$, if for all contexts-with-hole $\Phi_\circ$ such that the all variables declared in $\Phi_\circ$ are fresh for Γ, Δ and σ, the following two conditions hold.

1. For all types $\Phi(\Delta) \vdash A$ Type, we can derive $(\Phi_\circ[\sigma])(\Gamma) \vdash A[\sigma] = A[\tau]$ Type.
2. For all terms $\Phi(\Delta) \vdash M : A$, we can derive $(\Phi_\circ[\sigma])(\Gamma) \vdash M[\sigma] = M[\tau] : A[\sigma]$.

By this definition, two substitutions are equal if they give equal results in all situations. The requirement that $\Phi_\circ$ declares only fresh variables is necessary to ensure that both $(\Phi_\circ[\sigma])(\Gamma)$ and $\Phi(\Delta)$ are valid bunches and that names of temporary variables in σ (i.e. the variables in $FV(\sigma) \setminus (FV(\Gamma) \cup FV(\Delta))$) do not collide with the names of variables in $\Phi_\circ$. The requirement that $\Phi_\circ$ declares only fresh variables corresponds to the side-conditions on the rules (SUB-$*$) and (SUB-LIFT). An example where the requirement is important is for the 'garbage collection' equation $\langle x/z\rangle = \langle\rangle\colon (x\colon A) \to \diamond$. We can show this equation, but only because $\Phi_\circ$ can be assumed not to contain z.

Next we show that the representation of substitutions enjoys familiar properties of substitution, as in e.g. [110].

Lemma 5.5.3. *For any $\sigma\colon \Gamma \to \Delta$, the following are true.*

- *If* $\Delta \vdash A = B$ Type *then* $\Gamma \vdash A[\sigma] = B[\sigma]$ Type.
- *If* $\Delta \vdash M = N : A$ *then* $\Gamma \vdash M[\sigma] = N[\sigma] : A$.

Proof. Using the congruences (SUBST-TY-CGR), ($*$-E-TY-CGR), (SUBST-TM-CGR) and ($*$-E-CGR). □

Lemma 5.5.4. *Equality of substitution is an equivalence relation with the following properties.*

- *If* $\sigma = \sigma' \colon \Gamma \to \Delta$ *and* $\tau = \tau' \colon \Delta \to \Phi$ *then* $(\tau \circ \sigma) = (\tau' \circ \sigma') \colon \Gamma \to \Phi$.
- $\sigma = \sigma \circ \langle\rangle = \langle\rangle \circ \sigma \colon \Gamma \to \Delta$ *for all* $\sigma \colon \Gamma \to \Delta$.
- $\rho \circ (\tau \circ \sigma) = (\rho \circ \tau) \circ \sigma \colon \Gamma \to \Delta$ *for all* $\sigma \colon \Gamma \to \Phi$, $\tau \colon \Phi \to \Psi$ *and* $\rho \colon \Psi \to \Delta$.

Proof. By definition of the substitution action $(-)[\sigma]$ and using Lemma 5.5.3. □

Lemma 5.5.5 (Substitution Lemma)**.** *For any substitution* $\sigma \colon \Gamma \to \Delta$*, any term* $\Delta \vdash M : A$ *and any fresh variable* x*, we have* $\langle M/x\rangle \circ \sigma = \sigma \circ \langle M[\sigma]/x\rangle \colon \Gamma \to (\Delta, x \colon A)$.

Proof. We show the assertion by induction on the substitution σ.

- σ is $\langle\rangle$. The assertion holds trivially since $\mathscr{J}[\langle\rangle] = \mathscr{J}$.
- σ is $\sigma'' \circ \sigma'$. Let $\Delta \vdash M : A$ and let x be a fresh variable. It follows by inversion that there exists Φ such that $\sigma' \colon \Gamma \to \Phi$ and $\sigma'' \colon \Phi \to \Delta$ are derivable. Furthermore, we can assume that x is fresh not only for σ but also for Φ, because if neither σ' nor σ'' mention x, then any occurrence of x in Φ can be replaced with a fresh variable. The assertion then follows at once from the following diagram, both subsquares of which commute by induction hypothesis.

$$\begin{array}{ccccc} \Gamma & \xrightarrow{\sigma'} & \Phi & \xrightarrow{\sigma''} & \Delta \\ {\scriptstyle \langle M[\sigma''][\sigma']/x\rangle}\downarrow & & {\scriptstyle \langle M[\sigma'']/x\rangle}\downarrow & & \downarrow{\scriptstyle \langle M/x\rangle} \\ \Gamma, x \colon A[\sigma''][\sigma'] & \xrightarrow[\sigma']{} & \Phi, x \colon A[\sigma''] & \xrightarrow[\sigma'']{} & \Delta, x \colon A \end{array}$$

- σ is $\langle N/y\rangle$. This case follows from the normal syntactic substitution lemma that $\mathscr{J}[M/x][N/y] = \mathscr{J}[N/y][M[N/y]/x]$ holds whenever $x \notin FV(N)$, see e.g. [110, Chapter VIII].
- σ is $\langle N/u \colon C * v \colon D\rangle$. Let $\Delta \vdash M : A$ and let x be a fresh variable. The derivation of $\sigma \colon \Gamma \to \Delta$ must end in an application of (SUB-LET) followed by a number of applications of (SUB-$*$), (SUB-LIFT), (SUB-STR) and (SUB-BU-CONV). These rules can be permuted so that the derivation ends in a number of applications of (SUB-$*$) and (SUB-LIFT) followed by a number of applications of (SUB-STR) and (SUB-BU-CONV). This implies that there exist derivable judgements $\vdash \Gamma = \Gamma''$ Bunch, $\Gamma'' \succcurlyeq (\Phi_\circ[\langle N/u * v\rangle])(\Gamma')$ and $\Phi(u \colon C * v \colon D) \succcurlyeq \Delta$, and that we have $\Gamma' \vdash N : C * D$.

  We have to show $R[\langle M/x\rangle][\langle N/u \colon C * v \colon D\rangle] = R[\langle N/u \colon C * v \colon D\rangle][M[\langle N/u \colon C * v \colon D\rangle]/x]$ for all terms $\Psi(\Delta, x \colon A) \vdash R : B$, where $\Psi_\circ$ declares only fresh variables. From now on we omit the types C and D for readability. Consider the following derivation.

$$(*\text{-E}^+)\ \dfrac{\text{Lemma 4.2.2}\ \dfrac{(\text{CONG})\ \dfrac{\Delta \vdash M : A}{\Phi(u \colon C * v \colon D) \vdash M : A}}{\Psi(\Phi(u \colon C * v \colon D)) \vdash M : A}}{(\Psi(\Phi_\circ)[\langle z/u * v\rangle])(z \colon C * D) \vdash M[\langle z/u * v\rangle] : A[\langle z/u * v\rangle]}$$

We then have the following derivation, in which we use the abbreviations

$$\begin{aligned}
\Psi_1 &\stackrel{\text{def}}{=} \Psi(\Phi_\circ, x\colon A)[\langle z/u*v\rangle](z\colon C*D)\\
&= \Psi_\circ[\langle z/u*v\rangle](\Phi_\circ[\langle z/u*v\rangle](z\colon C*D), x\colon A[\langle z/u*v\rangle])\\
\Psi_2 &\stackrel{\text{def}}{=} \Psi(\Phi_\circ)[\langle z/u*v\rangle](z\colon C*D)[M[\langle z/u*v\rangle]/x]\\
\Psi_3 &\stackrel{\text{def}}{=} \Psi(\Phi_\circ)[\langle N/u*v\rangle](\Gamma')[M[\langle N/u*v\rangle]/x]\\
&= \Psi_\circ[\langle N/u*v\rangle](\Phi_\circ[\langle N/u*v\rangle](\Gamma'))[M[\langle N/u*v\rangle]/x]\\
\Psi_4 &\stackrel{\text{def}}{=} \Psi_\circ[\langle N/u*v\rangle](\Gamma'')[M[\langle N/u*v\rangle]/x] = \Psi_\circ[\langle N/u*v\rangle \circ \langle M[\langle N/u*v\rangle]/x\rangle](\Gamma'')\\
\Psi_5 &\stackrel{\text{def}}{=} \Psi_\circ[\langle N/u*v\rangle \circ \langle M[\langle N/u*v\rangle]/x\rangle](\Gamma)
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Psi(\Delta, x\colon A)\vdash R : B}{\Psi(\Phi(u\colon C*v\colon D), x\colon A)\vdash R : B}\,(\text{CONG})}{\Psi_1 \vdash R[\langle z/u*v\rangle] : B[\langle z/u*v\rangle]}\,(*\text{-E}^+)}{\Psi_2 \vdash (R[\langle z/u*v\rangle])[M[\langle z/u*v\rangle]/x] : B'}\,(\text{SUBST})}{\Psi_3 \vdash R[\langle N/u*v\rangle][M[\langle N/u*v\rangle]/x] = R[\langle u*v/u*v\rangle][M[\langle u*v/u*v\rangle]/x][\langle N/u*v\rangle] : B''}\,(*\text{-}\eta)}{\Psi_4 \vdash R[\langle N/u*v\rangle][M[\langle N/u*v\rangle]/x] = R[\langle u*v/u*v\rangle][M[\langle u*v/u*v\rangle]/x][\langle N/u*v\rangle] : B''}\,(\text{CONG})}{\Psi_5 \vdash R[\langle N/u*v\rangle][M[\langle N/u*v\rangle]/x] = R[\langle u*v/u*v\rangle][M[\langle u*v/u*v\rangle]/x][\langle N/u*v\rangle] : B''}\,(\text{BU-CONV})$$

Using $(*\text{-}\beta)$ and the congruence rules, this equation can be simplified to

$$R[\langle N/u*v\rangle][M[\langle N/u*v\rangle]/x] = (R[M/x])[\langle N/u*v\rangle].$$

A corresponding equation for the types can be derived similarly. Note that the context Ψ_5 is just as required in the definition of equality for substitutions. By definition of the substitution action $(-)[\sigma]$, this is just the required equation.

□

Lemma 5.5.6. *If* $\sigma\colon \Gamma \to \Delta$ *and* $\Delta \vdash B$ Type *and* x *is a fresh variable, then* $\Gamma, x\colon B[\sigma] \vdash x = x[\sigma] : B[\sigma]$.

Proof. The proof goes by induction on σ. The interesting case is where σ is $\langle N/u\colon C*v\colon D\rangle$. As in the previous lemma, it follows by inversion that there exist derivable judgements $\vdash \Gamma = \Gamma''$ Bunch, $\Gamma'' \succcurlyeq (\Phi_\circ[\langle N/u*v\rangle])(\Gamma')$ and $\Phi(u\colon C*v\colon D) \succcurlyeq \Delta$, and that we have $\Gamma' \vdash N : C*D$. The required equation can be derived as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Delta \vdash B \text{ Type}}{\Phi(u\colon C*v\colon D)\vdash B \text{ Type}}\,(\text{CONG})}{\Phi(u\colon C*v\colon D), x\colon B \vdash x : B}\,(\text{PROJ})}{(\Phi_\circ[\langle z/u*v\rangle])(z\colon C*D), x\colon B[\langle z/u*v\rangle] \vdash x : B[\langle z/u*v\rangle]}\,(*\text{-E}^+)}{(\Phi_\circ[\langle N/u*v\rangle])(\Gamma'), x\colon B[\langle N/u*v\rangle] \vdash x = x[\langle N/u*y\rangle] : B[\langle N/u*v\rangle]}\,(*\text{-}\eta)}{\Gamma, x\colon B[\langle N/u*v\rangle] \vdash x = x[\langle N/u*y\rangle] : B[\langle N/u*v\rangle]}\,(\text{CONG}),(\text{BU-CONV})$$

□

Lemma 5.5.7. *For all* $\sigma\colon \Gamma \to \Delta$ *and* $\tau\colon \Phi \to \Psi$ *that satisfy* $FV(\Gamma, \sigma, \Delta) \cap FV(\Phi, \tau, \Psi)$*, the equality of substitutions* $(\sigma \circ \tau) = (\tau \circ \sigma)\colon \Gamma * \Phi \to \Delta * \Psi$ *holds.*

Proof. Although this lemma is trivial for normal substitution, in the presence of let-terms it requires commuting conversions. The proof goes by induction on τ. In the case where τ is $\langle M/x\colon A * y\colon B\rangle$ we use $(*\text{-}\eta)$ and $(*\text{-}\beta)$ as in the previous lemmas to derive $(\text{let } M \text{ be } x*y \text{ in } (R[\sigma]) = (\text{let } M \text{ be } x*y \text{ in } R)[\sigma]$, which gives the required property. □

Lemma 5.5.8. *If we have* $\sigma = \tau\colon \Gamma \to \Delta$ *and* $\Delta \vdash A$ Type *and* x *is a fresh variable, then the equation* $\sigma = \tau\colon (\Gamma, x\colon A[\sigma]) \to (\Delta, x\colon A)$ *holds.*

Proof. Immediate from the definition of equality for substitution. □

Lemma 5.5.9. *If* $\tau\colon \Gamma \to (\Delta, x\colon A)$ *and* $\sigma\colon \Gamma \to \Delta$ *are such that* $\sigma = \tau\colon \Gamma \to \Delta$ *holds, then we have* $\tau = \langle z/x\rangle \circ \sigma \circ \langle x[\tau]/z\rangle\colon \Gamma \to (\Delta, x\colon A)$*, where* z *is some/any fresh variable.*

Proof. Let z be fresh. From $\sigma = \tau\colon \Gamma \to \Delta$ we get $\sigma = \tau\colon (\Gamma, z\colon A[\sigma]) \to (\Delta, z\colon A)$ by the previous lemma. Since $\langle x[\tau]/z\rangle\colon \Gamma \to (\Gamma, z\colon A[\sigma])$, this implies $\sigma \circ \langle x[\tau]/z\rangle = \tau \circ \langle x[\tau]/z\rangle\colon \Gamma \to (\Delta, z\colon A)$. By Lemma 5.5.5, we have $\langle x/z\rangle \circ \tau = \tau \circ \langle x[\tau]/z\rangle\colon \Gamma \to (\Delta, x\colon A, z\colon A)$, which implies $\langle x/z\rangle \circ \tau = \tau \circ \langle x[\tau]/z\rangle\colon \Gamma \to (\Delta, z\colon A)$. Using transitivity, we get $\sigma \circ \langle x[\tau]/z\rangle = \langle x/z\rangle \circ \tau\colon \Gamma \to (\Delta, z\colon A)$. By composition, we have $\langle z/x\rangle \circ \sigma \circ \langle x[\tau]/z\rangle = \langle z/x\rangle \circ \langle x/z\rangle \circ \tau\colon \Gamma \to (\Delta, x\colon A)$, which gives the required $\langle z/x\rangle \circ \sigma \circ \langle x[\tau]/z\rangle = \tau\colon \Gamma \to (\Delta, x\colon A)$. □

5.6 Internalising Contexts and Display Property

In the presence of $*$-types, Σ-types and 1-types, each context can be represented by a type. The representation is induced by the following isomorphisms.

$$\diamond \underset{\langle\rangle}{\overset{\langle \text{unit}/z\rangle}{\rightleftarrows}} (z\colon 1) \qquad (x\colon A, y\colon B) \underset{\langle \text{fst}(z)/x\rangle \circ \langle \text{snd}(z)/y\rangle}{\overset{\langle\langle x,y\rangle/z\rangle}{\rightleftarrows}} (z\colon \Sigma x\colon A.B) \qquad (x\colon A) * (y\colon B) \underset{\langle z/x*y\rangle}{\overset{\langle x*y/z\rangle}{\rightleftarrows}} (z\colon A*B)$$

Making repeated use of these isomorphisms, each context Γ can be represented as a context declaring a single variable. The isomorphisms are used inside-out, so that, for instance, to represent the context $(x\colon A) * (\Delta, y\colon B)$ we first obtain an isomorphism $\sigma\colon (z\colon \lfloor\Delta\rfloor) \cong \Delta$, which we then lift to an isomorphism between $(x\colon A) * (\Delta, y\colon B)$ and $(x\colon A) * (z\colon \lfloor\Delta\rfloor, y\colon B[\sigma])$. It should be clear how to use the above isomorphisms to internalise the resulting context.

Formally, we define for each context Γ a type $\lfloor\Gamma\rfloor$ and a substitution $\mathsf{o}(z,\Gamma)\colon (z\colon \lfloor\Gamma\rfloor) \to \Gamma$ simultaneously by induction on the structure of Γ.

$$\begin{aligned} \lfloor\diamond\rfloor &= 1 & \mathsf{o}(z,\diamond) &= \langle\rangle \\ \lfloor\Gamma * \Delta\rfloor &= \lfloor\Gamma\rfloor * \lfloor\Delta\rfloor & \mathsf{o}(z,\Gamma*\Delta) &= \mathsf{o}(u,\Gamma) \circ \mathsf{o}(v,\Delta) \circ \langle z/u\colon \lfloor\Gamma\rfloor * v\colon \lfloor\Delta\rfloor\rangle \\ \lfloor\Gamma, x\colon A\rfloor &= \Sigma z\colon \lfloor\Gamma\rfloor.A[\mathsf{o}(z,\Gamma)] & \mathsf{o}(z,(\Gamma, x\colon A)) &= \mathsf{o}(u,\Gamma) \circ \langle \text{fst}(z)/u\rangle \circ \langle \text{snd}(z)/x\rangle \end{aligned}$$

In these definitions, u, v and z are arbitrary fresh variables. The choice of fresh variables is irrelevant with respect to equality of substitutions.

To give an example, we internalise the context in $((x\colon A * y\colon B), u\colon C) * (v\colon D) \vdash E$ Type:

$$\dfrac{\dfrac{\dfrac{((x\colon A * y\colon B), u\colon C) * (v\colon D) \vdash E \text{ Type}}{(a\colon A{*}B, u\colon C[\langle a/x * y\rangle]) * (v\colon D) \vdash E[\langle a/x * y\rangle] \text{ Type}}}{b\colon (\Sigma a\colon A{*}B.C[\langle a/x * y\rangle]) * (v\colon D) \vdash E[\langle \mathrm{fst}(b)/x * y\rangle][\mathrm{snd}(b)/u] \text{ Type}}}{c\colon ((\Sigma a\colon A{*}B.(C[\langle a/x * y\rangle])) * D) \vdash E[\langle \mathrm{fst}(b)/x * y\rangle][\mathrm{snd}(b)/u][\langle c/b * v\rangle] \text{ Type}}$$

To define an inverse of $\mathsf{o}(z,\Gamma)$, we define a term $\Gamma \vdash \mathsf{i}(\Gamma) : \lfloor\Gamma\rfloor$ satisfying

$$\Gamma \vdash A[\mathsf{o}(z,\Gamma)]\,[\mathsf{i}(\Gamma)/z] = A \text{ Type} \qquad z\colon \lfloor\Gamma\rfloor \vdash B\,[\mathsf{i}(\Gamma)/z][\mathsf{o}(z,\Gamma)] = B \text{ Type}$$
$$\Gamma \vdash M[\mathsf{o}(z,\Gamma)]\,[\mathsf{i}(\Gamma)/z] = M : A \qquad z\colon \lfloor\Gamma\rfloor \vdash N\,[\mathsf{i}(\Gamma)/z][\mathsf{o}(z,\Gamma)] = N : B$$

for all types $\Gamma \vdash A$ Type and $z\colon \lfloor\Gamma\rfloor \vdash B$ Type, all terms $\Gamma \vdash M : A$ and $z\colon \lfloor\Gamma\rfloor \vdash N : B$, and all fresh variables z. The definition is given inductively by:

$$\begin{aligned}\mathsf{i}(\diamond) &= \mathrm{unit}\\ \mathsf{i}(\Gamma * \Delta) &= \mathrm{pair}^{*}_{\lfloor\Gamma\rfloor,\lfloor\Delta\rfloor}(\mathsf{i}(\Gamma),\mathsf{i}(\Delta))\\ \mathsf{i}(\Gamma, x\colon A) &= \mathrm{pair}_{(z:\lfloor\Gamma\rfloor)A[\mathsf{o}(z,\Gamma)]}(\mathsf{i}(\Gamma),x)\end{aligned}$$

Notice that for well-typedness in the last case we need the equality $A[\mathsf{o}(z,\Gamma)]\,[\mathsf{i}(\Gamma)/z] = A$, given by the induction hypothesis. The verifications are routine.

Proposition 5.6.1 (Display Property)**.** *For each type $\Gamma(\Delta) \vdash A$ Type there exists a type $\Delta \vdash B$ Type such that there is a one-to-one correspondence between terms $\Gamma(\Delta) \vdash M : A$ and $\Delta \vdash N : B$. Precisely, this means that, for each term $\Gamma(\Delta) \vdash M : A$, there exists a term M_Γ such that $\Delta \vdash M_\Gamma : B$ is derivable, and, for each term $\Delta \vdash N : B$, there exists a term N^Γ such that $\Gamma(\Delta) \vdash N^\Gamma : A$ is derivable. Moreover, these assignments are such that $\Gamma(\Delta) \vdash M = (M_\Gamma)^\Gamma : A$ and $\Delta \vdash N = (N^\Gamma)_\Gamma : B$ hold for all terms $\Gamma(\Delta) \vdash M : A$ and $\Delta \vdash N : B$.*

Proof. By induction on $\Gamma_\circ$, using internalisation together with Π and Π^*-types. The base case where $\Gamma_\circ$ is $\bigcirc$ follows trivially by letting B be A. If $\Gamma_\circ$ is $\Gamma'_\circ, x\colon C$ then we can derive $\Gamma'(\Delta) \vdash \Pi x\colon C.A$ Type. By the equations for Π-types, the assignment $M \mapsto \lambda x\colon C.M$ gives a one-to-one correspondence between terms of type $\Gamma(\Delta) \vdash A$ Type and terms of type $\Gamma'(\Delta) \vdash \Pi x\colon C.A$ Type. Its inverse is application $N \mapsto N\,x$. The assertion follows by using the induction hypothesis for $\Gamma'(\Delta) \vdash \Pi x\colon C.A$ Type. If $\Gamma_\circ$ is $\Gamma'_\circ * \Phi$ then we can derive $\Gamma'(\Delta) * z\colon \lfloor\Phi\rfloor \vdash A[\mathsf{o}(z,\Phi)]$ Type and further $\Gamma'(\Delta) \vdash \Pi^* z\colon \lfloor\Phi\rfloor.A[\mathsf{o}(z,\Phi)]$ Type. The assignment $M \mapsto \lambda^* z\colon \lfloor\Phi\rfloor.M[\mathsf{o}(z,\Phi)]$ is then a one-to-one correspondence between terms of type A and terms of type $\Pi^* z\colon \lfloor\Phi\rfloor.A[\mathsf{o}(z,\Phi)]$. Its inverse is $N \mapsto N@\mathsf{i}(\Phi)$. The assertion follows by using the induction hypothesis for $\Gamma'(\Delta) \vdash \Pi^* z\colon \lfloor\Phi\rfloor.A[\mathsf{o}(z,\Phi)]$ Type. Since, using this argument and (SWAP), the case where Γ is $\Phi * \Gamma'_\circ$ follows as well, this completes the proof. □

Using the above proposition, we can omit parameter contexts in the formulation of elimination rules, as discussed in Section 4.3. We are therefore in essentially the same situation as in additive type theory, where elimination rules are also usually formulated without parameter contexts.

5.7 Strong Π^*-types

The rules for Π^*-types that we have given so far are sound but may not be complete. The source of incompleteness is the definition of strong simple monoidal products Π^* (Definition 2.5.12), which relates two terms of the form $(M_1@x)[\sigma_1]$ and $(M_2@x)[\sigma_2]$ for arbitrary substitutions σ_1 and σ_2. The congruence rule (Π^*-E-CGR), however, does not account for substitutions of the form $\langle N/u*v\rangle$. In order for $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ to be complete, we strengthen the rule (Π^*-E-CGR).

$$(\Pi^*\text{-E-CGR}^+)\ \frac{\begin{array}{ll} & \vdash A_1 = A_2\ \text{Type} \\ \Delta_1 \vdash M_1 : \Pi^* x\colon A_1.B_1 & \Gamma \vdash (\Pi^* x\colon A_1.B_1)[\sigma_1] = (\Pi^* x\colon A_2.B_2)[\sigma_2]\ \text{Type} \\ \Delta_2 \vdash M_2 : \Pi^* x\colon A_2.B_2 & \Gamma \vdash B_1[\sigma_1] = B_2[\sigma_2]\ \text{Type} \\ \sigma_1 : \Gamma \to (\Delta_1 * x\colon A_1) & \Gamma \vdash M_1[\sigma_1] = M_2[\sigma_2] : (\Pi^* x\colon A_1.B_1)[\sigma_1] \\ \sigma_2 : \Gamma \to (\Delta_2 * x\colon A_2) & \Gamma \vdash x[\sigma_1] = x[\sigma_2] : A_1 \end{array}}{\Gamma \vdash (\mathrm{app}^*_{(x:A_1)B_1}(M_1,x))[\sigma_1] = (\mathrm{app}^*_{(x:A_2)B_2}(M_2,x))[\sigma_2] : B_1[\sigma_1]}$$

That (Π^*-E-CGR) derives from (Π^*-E-CGR$^+$) can be seen by taking $\sigma_1 = \langle N_1/x\rangle$ and $\sigma_2 = \langle N_2/x\rangle$.

5.8 Further Work

The most obvious omission from this chapter is an algorithm for type-checking. As usual in dependent type theories, the main work for giving such an algorithm goes into showing decidability of definitional equality. Deciding equality for $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ is complicated by the presence of let-terms and their commuting conversions. It may, however, be possible to exploit the extra assumption that $*$ is strict affine to avoid commuting conversions in an algorithm. In the next chapter we will show that it is sound to replace let-terms with the affine projections, that is to translate $(\text{let } M \text{ be } x{*}y \text{ in } N)$ to $N[\dot\pi_1(M)/x][\dot\pi_2(M)/y]$. This translation identifies, in particular, any two terms that are the same up to commuting conversion. In 7.2.2 we show that any two terms having the same translation are provably equal. This suggests that it may be easier to decide equality on the translated terms with affine projections. Perhaps, it is even possible to take as equations simply $\dot\pi_1(M{*}N) = M$, $\dot\pi_2(M{*}N) = N$ and $R = \dot\pi_1(R) * \dot\pi_2(R)$, where R is of type $A{*}B$. An algorithm based on these equations should be no more complicated than an algorithm for $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$. It would evidently be complete, i.e. it would be able to show all equations. But would it be sound?

Chapter 6

Interpretation and Soundness

In this chapter we define the semantic interpretation of the type theory from the previous chapter and show soundness. The main difficulty in doing this is how to deal with the multiplicative types $*$ and Π^*. The additive types, on the other hand, present no more problems than in the interpretation of standard dependent type theory. We therefore consider only Π-types in detail, leaving out 1-types and Σ-types for brevity. Hence, in this chapter we define the interpretation for $\mathbf{BT}(*, \Pi, \Pi^*)$, for which write just $\mathbf{BT}$.

6.1 Semantic Structure

There are well-known coherence problems with the interpretation of dependent types in, say, comprehension categories [46]. The problem is that the substitution laws given by Beck-Chevalley conditions, such as $\sigma^*(\Pi_A B) \cong (\Pi_{\sigma^* A}\{\overline{\sigma}\}^* B)$, hold only up to isomorphism and not equality. As a result one has to solve coherence problems for the interpretation of substitution. Curien shows how to treat 'substitution up to isomorphism' [26]. A simpler solution is to require the isomorphism $\sigma^*(\Pi_A B) \cong (\Pi_{\sigma^* A}\{\overline{\sigma}\}^* B)$ in the semantics to be the identity, i.e to work with split fibrations. This is the solution most often adopted in the literature, see e.g. [105, 48, 85]. By considering split fibrations, the coherence problem is shifted to the semantics, where we now have to find split fibrations for the models we are interested in. In Section 3.5.2, we have shown how to do this for our models with names.

We interpret $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$ in the following structure.

Definition 6.1.1. A $(*, 1, \Sigma, \Pi, \Pi^*)$-*type-category* is given by the following data.

1. A split closed comprehension category with unit $p\colon \mathbb{E} \to \mathbb{B}$.
2. A strict affine monoidal structure $*$ on $\mathbb{B}$, together with, for any two objects A and B in $\mathbb{E}_1$, an object $A \overline{*} B$ in $\mathbb{E}_1$ and an isomorphism $l_{A,B}\colon \{A\} * \{B\} \to \{A \overline{*} B\}$ in $\mathbb{B}$.
3. Strong split simple monoidal products Π^* with respect to the monoidal structure $*$.

6.2 Overview

Although the interpretation and the proof of soundness are essentially straightforward, the technical development is rather lengthy. This is already the case for normal dependent types. Streicher's detailed proof of soundness for the calculus of constructions takes up 50 pages in [105]. Therefore, before we dive into the details, we give a brief overview of the interpretation.

An obvious first idea to define the interpretation of **BT** is by induction on the derivations. This, however, does not readily work with dependent types. To see the problem, consider the application rule.

$$(\Pi\text{-E})\ \frac{\Gamma, x\colon A \vdash B\ \mathrm{Type} \qquad \Gamma \vdash M : \Pi x\colon A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathrm{app}_{(x:A)B}(M,N) : B[N/x]}$$

The idea would be to use the induction hypothesis to obtain interpretations of the premises and then interpret the term in the conclusion by the map $N^*\varepsilon_{A,B} \circ M\colon 1 \to \Pi_A B = N^*\pi_A^*\Pi_A B \to N^*B$ in $\mathbb{E}_\Gamma$, where ε_A is the counit of $\pi_A^* \dashv \Pi_A$. There are at least to problems with this idea. First, because the type theory allows different derivations of the same judgement, there is, a priori, no reason why the A in the type of M should have received the same interpretation as the type of N. But as the argument types might not match, we cannot use the semantic application. One possible solution to this problem for normal dependent type theory is to restrict the rules so that each judgement has effectively only one derivation. For **BT** it is not clear how to formulate the rules in such a way, for instance because weakening is not admissible. The second problem is that we must relate substitution in the syntax to reindexing in the semantics. In order for $N^*\varepsilon_{A,B} \circ M$ to be an interpretation for the conclusion of the above rule, its codomain N^*B must be the interpretation of the type $B[N/x]$, but this information is not available.

A possible solution to these problems is to prove that the interpretations of any two derivations of the same judgement are in fact the same. This is the approach we take in this chapter. It addresses the second of the above issues because the substitution rule is interpreted by semantic reindexing. Notice that, because of the problems illustrated with application above, we cannot easily define the interpretation of derivations first and then show that two derivations of the same judgement are equal.

As a way of working with the semantics, we first introduce a dependent type theory **ES** that corresponds directly to the semantics. Since substitution and the structural rules correspond to uses of reindexing functors in the comprehension category, a type theory corresponding to the semantics is naturally given by an explicit substitution type theory with de Bruijn indices. The treatment of the semantics as an explicit substitution calculus is folklore. For example, Cartmell's categories with families [18, 105, 48] may be viewed as a formulation of type theory with explicit substitutions and de Bruijn indices; other examples can be found in [26, 93]. We mention here just the treatment of $*$-types in the explicit substitution calculus. For $*$-types we have an isomorphism $l_{A,B}\colon \{A\} * \{B\} \to \{A \overline{*} B\}$. In suggestive notation, this morphism corresponds to the substitution $\langle x*y/z\rangle\colon (x\colon A) * (y\colon B) \to (z\colon A*B)$. Its inverse, which we denote by $\langle z/x*y\rangle$, corresponds to the let-term for $*$-elimination, see also the previous chapter. Thus, the let-terms in the $*$-elimination become a special case of substitution. In the next section, we formulate the type theory **ES** with explicit substitutions and give a sound semantic interpretation for it.

Having defined the explicit substitution type theory **ES** for working with the semantics, the problem of defining the semantic interpretation of **BT** can be reduced to interpreting **BT** in **ES**. This means that we would like to translate derivations in **BT** to derivations in **ES**. Of course, this translation should be reasonable, so that the constructions in **BT** are mapped to the corresponding constructions in **ES**, e.g that Π-types in **BT** are mapped to Π-types in **ES**. From any judgement K with explicit substitutions in **ES** we can obtain a judgement J in **BT**, essentially by replacing the explicit substitutions with actual syntactic substitutions. Let us write $K \Downarrow J$ if J is obtained from K in this way. That the translation from **BT** to **ES** is reasonable then means that a derivation of a judgement J in **BT** is translated to a derivation of a judgement K in **ES** satisfying $K \Downarrow J$. In defining such a translation, we encounter the problem outlined above, that we must show that two derivations of the same judgement J in **BT** receive provably equal interpretations in **ES**. To show this it suffices to prove that $K' \Downarrow J$ and $K'' \Downarrow J$ implies $K' = K''$ in **ES**.

Showing that $K' \Downarrow J$ and $K'' \Downarrow J$ implies $K' = K''$ in **ES** takes up most of this chapter. However, the idea is quite straightforward. The proof goes by induction on the combined size of K' and K''. For example, consider the induction case for Π-types. If $K' \Downarrow \Pi x\colon A.B$ and $K'' \Downarrow \Pi x\colon A.B$ then K' and K'' can be assumed to be of the form $(\Pi A'.B')[\sigma'']$ and $(\Pi A''.B'')[\sigma'']$ respectively, such that $A'[\sigma'] \Downarrow A$ and $A''[\sigma''] \Downarrow A$ and $B'[\sigma'_\bullet] \Downarrow B$ and $B''[\sigma''_\bullet] \Downarrow B$, where $(-)_\bullet$ denotes lifting of substitutions in a larger context. All these types are smaller, so that we can use the induction hypothesis to obtain $A'[\sigma'] = A''[\sigma'']$ and $B'[\sigma'_\bullet] = B''[\sigma''_\bullet]$. By the congruence rule and the substitution equation for Π-types, we can conclude $K' = K''$. A second case worth mentioning is that where we have $K' \Downarrow \mathrm{pair}^*_{A,B}(M,N)$ and $K'' \Downarrow \mathrm{pair}^*_{A,B}(M,N)$. In this case, K' and K'' can be assumed to be of the form $(\mathrm{pair}^*_{A',B'}(M',N'))[\sigma']$ and $(\mathrm{pair}^*_{A'',B''}(M'',N''))[\sigma'']$ respectively. To prove $(\mathrm{pair}^*_{A',B'}(M',N'))[\sigma'] = (\mathrm{pair}^*_{A'',B''}(M'',N''))[\sigma'']$ we make essential use of a version of the rule (INJECT). By this rule it is enough to prove $M'[\dot\pi'_1][\sigma'] = M''[\dot\pi''_1][\sigma'']$ and $N'[\dot\pi'_2][\sigma'] = N''[\dot\pi''_2][\sigma'']$, where $\dot\pi'_1$, $\dot\pi''_1$, $\dot\pi'_2$ and $\dot\pi''_2$ are weakenings used to bring the terms in the right context (Note that M' and M'' could occur in different contexts). These equations can be proved by the induction hypothesis, since the fact that both K' and K'' translate to $\mathrm{pair}^*_{A,B}(M,N)$ allows us to assume that $M'[\dot\pi'_1][\sigma']$ and $M''[\dot\pi''_1][\sigma'']$ both translate to M, and analogue facts hold for A, B and N. We point out that for this argument to work we need the assumption that the monoidal structure $*$ is strict affine. Just like in these two cases, it can be seen that all cases for term constructors amount to pushing the substitutions σ' and σ'' inside the terms. Note that we consider let-terms as substitutions, so that lets are being pushed inside the terms as well. In the end we have to handle the case of variables. Since substitutions contain lets, we have to consider variables under a number of lets. Consider, for simplicity, just the special case where K' and K'' translate to a term of the form $(\mathrm{let}\ M\ \mathrm{be}\ x{:}A{*}y{:}B\ \mathrm{in}\ z)$. Because z is a variable, we can assume that M is typeable in the current context and not just in some not uniquely determined sub-context. Then we can use the induction hypothesis for M to show the equality $K' = K''$.

For the additive type formers, this argument is implicitly contained in the Streicher's proof of soundness for the calculus of constructions. In [105], Streicher essentially shows that each derivation of a type/term in **ES** can be brought in a normal form having the same interpretation. In the normal forms

the only kinds of substitutions are weakenings and these weakenings occur only on variables. It follows from this that $K' \Downarrow J$ and $K'' \Downarrow J$ implies $K' = K''$. Note that in our case such normal forms need not exist, for example because weakening is not admissible.

Already from the rough proof sketch above, it can be seen that the only kind of let-term we have to deal with are of the form $(\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } z)$. If z is different from both x and y then this term equals z, i.e. we can remove the let. The other two cases $(\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } x)$ and $(\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } y)$ are just the affine projections $A * B \to A$ and $A * B \to B$. Therefore, it is technically convenient to remove the let-terms altogether from **BT** and replace them with the two projections.

6.3 Let-free Syntax

The syntax of $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$ contains let-terms for the elimination of $*$-types. As discussed in the previous chapter, the choice of let-terms is motivated by syntactic considerations. From a semantic point of view, however, let-terms contain more information than necessary. For instance, a commuting conversion should correspond to an equality in the semantics. For the semantic interpretation, it is more convenient use the two projections $A * B \to A$ and $A * B \to B$ as eliminations for $*$-types.

We now introduce the let-free syntax. It differs from the syntax of $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$ only in that let-terms for $*$-types are replaced with two projections $l_{A,B}(M)$ and $r_{A,B}(M)$.

$$
\begin{array}{rl}
\text{Contexts} & \Gamma ::= \diamond \mid \Gamma * \Gamma \mid \Gamma, x\colon A \\
\text{Types} & A ::= T(M_1, \ldots, M_n) \mid 1 \mid A{*}A \mid \Sigma x\colon A.A \mid \Pi x\colon A.A \mid \Pi^* x\colon A.A \\
\text{Terms} & M ::= x \mid f(M_1, \ldots, M_n) \mid \text{unit} \\
& \quad \mid \text{pair}^*_{A,A}(M,M) \mid l_{A,A}(M) \mid r_{A,A}(M) \\
& \quad \mid \text{pair}_{(x:A)A}(M,M) \mid \text{fst}_{(x:A)A}(M) \mid \text{snd}_{(x:A)A}(M) \\
& \quad \mid \lambda x\colon A.M \mid \text{app}_{(x:A)B}(M,M) \\
& \quad \mid \lambda^* x\colon A.M \mid \text{app}^*_{(x:A)B}(M,M)
\end{array}
$$

We identify terms up to renaming of bound variables.

There is an evident translation $|-|$ from the syntax of $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$ to the let-free syntax, in which each let is replaced by the two projections.

$$
\begin{aligned}
|T(M_1, \ldots, M_n)| &= T(|M_1|, \ldots, |M_n|) \\
|1| &= 1 \\
|A * B| &= |A| * |B| \\
|\Pi x\colon A.B| &= \Pi x\colon |A|.|B| \\
|\Sigma x\colon A.B| &= \Sigma x\colon |A|.|B| \\
|\Pi^* x\colon A.B| &= \Pi^* x\colon |A|.|B|
\end{aligned}
$$

$$
\begin{aligned}
|x| &= x \\
|f(M_1,\ldots,M_n)| &= f(|M_1|,\ldots,|M_n|) \\
|\mathrm{unit}| &= \mathrm{unit} \\
|\mathrm{pair}^*_{A,B}(M,N)| &= \mathrm{pair}^*_{|A|,|B|}(|M|,|N|) \\
|(M * N)| &= |M| * |N| \\
|\mathrm{pair}_{(x:A)B}(M,N)| &= \mathrm{pair}_{(x:|A|)|B|}(|M|,|N|) \\
|\mathrm{fst}_{(x:A)B}(M)| &= \mathrm{fst}_{(x:|A|)|B|}(|M|) \\
|\mathrm{snd}_{(x:A)B}(M)| &= \mathrm{snd}_{(x:|A|)|B|}(|M|) \\
|\lambda x\colon A.N| &= \lambda x\colon |A|.|N| \\
|\mathrm{app}_{(x:A)B}(M,N)| &= \mathrm{app}_{(x:|A|)|B|}(|M|,|N|) \\
|\lambda^* x\colon A.N| &= \lambda^* x\colon |A|.|N| \\
|\mathrm{app}^*_{(x:A)B}(M,N)| &= \mathrm{app}^*_{(x:|A|)|B|}(|M|,|N|) \\
|\text{let } M \text{ be } x{:}A * y{:}B \text{ in } N| &= |N|\,[l_{A,B}(|M|)/x]\,[r_{A,B}(|M|)/y]
\end{aligned}
$$

Note in particular that two terms M and N in $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ that differ only up to commuting conversion translate to equal terms under $|-|$.

To each term M in the explicit substitution calculus, we assign a term in the let-free syntax, essentially by replacing explicit substitutions with actual syntactic substitutions. In order to define this assignment, we need a representation of simultaneous substitution for the let-free syntax. We follow Stoughton [104] in our treatment of simultaneous substitution. A substitution χ is a function from variables to let-free terms. We use χ and ξ to range over simultaneous substitutions. We write *id* for the identity substitution.

A special case of substitutions are renamings, which are functions from variables to variables. We use α and β to range over renamings.

Given a set of variables V and substitutions χ and ξ, we use the notation $\chi\{\xi/V\}$ for the substitution defined by:

$$
(\chi\{\xi/V\})(x) = \begin{cases} \xi(x) & \text{if } x \in V, \\ \chi(x) & \text{otherwise.} \end{cases}
$$

For a term M, a variable x and a substitution χ, we write $\chi\{M/x\}$ for the substitution defined by:

$$
(\chi\{M/x\})(y) = \begin{cases} M & \text{if } x = y, \\ \chi(y) & \text{otherwise.} \end{cases}
$$

We abbreviate $id\{\xi/V\}$ and $id\{M/x\}$ by $\{\xi/V\}$ and $\{M/x\}$ respectively.

Given a set of variables V, we define a binary relation $=_V$ on substitutions as follows.

$$
\chi =_V \xi \quad \Longleftrightarrow \quad \forall x \in V.\, \chi(x) = \xi(x) \tag{6.1}
$$

The action of a substitution χ on types and terms, denoted by $A\ \chi$ and $M\ \chi$ respectively, is defined inductively by the following equations.

$$
\begin{aligned}
T(M_1,\ldots,M_n)\ \chi &= T(M_1\ \chi,\ldots,M_n\ \chi) \\
1\ \chi &= 1 \\
(A * B)\ \chi &= (A\ \chi) * (B\ \chi) \\
(\Pi x\colon A.B)\ \chi &= \Pi y\colon (A\ \chi).(B\ \chi\{y/x\}) && y \text{ fresh} \\
(\Sigma x\colon A.B)\ \chi &= \Sigma y\colon (A\ \chi).(B\ \chi\{y/x\}) && y \text{ fresh} \\
(\Pi^* x\colon A.B)\ \chi &= \Pi^* y\colon (A\ \chi).(B\ \chi\{y/x\}) && y \text{ fresh} \\
x\ \chi &= \chi(x) \\
f(M_1,\ldots,M_n)\ \chi &= f(M_1\ \chi,\ldots,M_n\ \chi) \\
\mathrm{unit}\ \chi &= \mathrm{unit} \\
\mathrm{pair}^*_{A,B}(M,N)\ \chi &= \mathrm{pair}^*_{A\ \chi,B\ \chi}(M\ \chi,N\ \chi) \\
l_{A,B}(M)\ \chi &= l_{A\ \chi,B\ \chi}(M\ \chi) \\
r_{A,B}(M)\ \chi &= r_{A\ \chi,B\ \chi}(M\ \chi) \\
\mathrm{pair}_{(x:A)B}(M,N)\ \chi &= \mathrm{pair}_{(y:(A\ \chi))B\ \chi\{y/x\}}(M\ \chi,N\ \chi) && y \text{ fresh} \\
\mathrm{fst}_{(x:A)B}(M)\ \chi &= \mathrm{fst}_{(y:(A\ \chi))(B\ \chi\{y/x\})}(M\ \chi) && y \text{ fresh} \\
\mathrm{snd}_{(x:A)B}(M)\ \chi &= \mathrm{snd}_{(y:(A\ \chi))(B\ \chi\{y/x\})}(M\ \chi) && y \text{ fresh} \\
(\lambda x\colon A.M)\ \chi &= \lambda y\colon (A\ \chi).(M\ \chi\{y/x\}) && y \text{ fresh} \\
\mathrm{app}_{(x:A)B}(M,N)\ \chi &= \mathrm{app}_{(y:(A\ \chi))(B\ \chi\{y/x\})}(M\ \chi,N\ \chi) && y \text{ fresh} \\
(\lambda^* x\colon A.M)\ \chi &= \lambda^* y\colon (A\ \chi).(M\ \chi\{y/x\}) && y \text{ fresh} \\
\mathrm{app}^*_{(x:A)B}(M,N)\ \chi &= \mathrm{app}^*_{(y:(A\ \chi))(B\ \chi\{y/x\})}(M\ \chi,N\ \chi) && y \text{ fresh}
\end{aligned}
$$

Because types and terms are identified up to renaming of bound variables, the choice of the fresh variable y in the above definition does not affect the result.

Composition of substitution is defined by $(\chi \circ \xi)(x) = \chi(x)\ \xi$. Note that this composition is the other way from that in [104]. In particular, for renamings α and β, we have $(\alpha \circ \beta)(x) = \beta(\alpha(x))$. We take this definition in order to agree with composition in the semantics and such that we have $J\ (\chi \circ \xi) = (J\ \chi)\ \xi$. Our definition of composition agrees with that in [49].

Lemma 6.3.1.

1. *For all substitutions χ and all name sets V, we have $id \circ \chi =_V \chi =_V id \circ \chi$.*
2. *For all substitutions χ_1, χ_2, χ_3 and all name sets V, we have $\chi_3 \circ (\chi_2 \circ \chi_1) =_V (\chi_3 \circ \chi_2) \circ \chi_1$.*
3. *For a bijective renaming α, we have $(\Pi^* x\colon A.B)\ \alpha = \Pi^* \alpha(x)\colon (A\ \alpha).(B\ \alpha)$. An analogous property holds for all binders.*

A proof is straightforward.

6.4 An Intermediate System with Explicit Substitutions

In this section we introduce **ES**, the explicit substitution calculus with de Bruijn indices that corresponds directly to the semantics. Since the purpose of introducing this calculus is as a vehicle for proving soundness of $\mathbf{BT}(*,\Pi,\Pi^*)$, we formulate the rules of inference for **ES** so that they already contain a translation to the let-free syntax.

The syntax for **ES** is given as follows.

$$\begin{array}{rl}
\text{Contexts} & \Gamma ::= \diamond \mid \Gamma * \Gamma \mid \Gamma, x\colon A \\
\text{Types} & A ::= T(\sigma) \mid A * A \mid \Pi A.A \mid \Pi^* A.A \\
\text{Terms} & M ::= 1 \mid f(\sigma) \mid \mathrm{pair}^*_{A,A}(M,M) \mid \lambda A.M \mid \mathrm{app}_{[A]A}(M,M) \mid \lambda^* A.M \mid \mathrm{app}^*_{[A]A}(M,M) \\
\text{Substitutions} & \sigma ::= \tau \mid \sigma \circ \sigma \mid \sigma * \sigma \mid \sigma_\bullet \mid \langle M \rangle \mid \langle M/A * A \rangle \\
\text{Structural maps} & \tau ::= id \mid \tau \circ \tau \mid \tau * \tau \mid \tau_\bullet \mid \mathrm{w} \mid \mathrm{unit} \mid \mathrm{unit}^{-1} \mid \mathrm{swap} \mid \mathrm{assoc} \mid \mathrm{assoc}^{-1}
\end{array}$$

We include variable names in the contexts. This is in order to simplify the translation of the explicit calculus to the let-free syntax and in order to be able to state concisely a coherence lemma for structural morphisms (Lemma 6.5.3). We write $FV(\Gamma)$ for the set of variables in Γ and $v(\Gamma)$ for the ordered list of variables in Γ. For brevity, we will often omit FV, e.g. writing just $\Gamma \cap \Delta$ for $FV(\Gamma) \cap FV(\Delta)$. As for **BT**, we make the convention that no variable may be declared more than once in a context.

We define dependently typed algebraic theories $\mathscr{T}$ in **ES** just as in Definition 4.1.1, only now using the above explicit substitution syntax.

The judgements for **ES** are the kinds judgements of $\mathbf{BT}(*,\Pi,\Pi^*)$ as well as judgements for substitutions and structural maps. Moreover, the judgements contain information about the syntactic translation, where we write $J \Downarrow K$ to mean that the statement J in the explicit substitution syntax translates to K in the let-free syntax. Thus, in the following judgements, Γ, Δ, A, B, M, N, σ and τ are formed in the explicit substitution syntax, while A', M' and χ are formed in the let-free syntax.

$\vdash \Gamma$ Bunch	The context Γ is well-formed.
$\Gamma \vdash A \Downarrow A'$ Type	The type A is well-formed in context Γ and A translates to A'.
$\Gamma \vdash M \Downarrow M' : A$	The term M has type A in context Γ and M translates to M'.
$\vdash \Gamma = \Delta$ Bunch	The contexts Γ and Δ are equal.
$\Gamma \vdash A = B$ Type	The types A and B in context Γ are equal.
$\Gamma \vdash M = N : A$	The terms M and N of type A in context Γ are equal.
$\sigma \Downarrow \chi\colon \Gamma \to \Delta$	σ is a substitution from Γ to Δ and σ translates to χ.
$\tau\colon \Gamma \Rightarrow \Delta$	τ is a structural map from Γ to Δ.
$\sigma = \tau\colon \Gamma \to \Delta$	σ and τ are equal substitutions from Γ to Δ

In the rest of this chapter, unless otherwise stated, all judgements are in the explicit substitution system **ES**. When we are not interested in the syntactic translation we will omit it from the judgements, e.g. writing $\Gamma \vdash A$ Type for $\Gamma \vdash A \Downarrow A'$ Type.

We formulate the rules using the following notation: $\mathscr{J}$ can be one of (A Type), ($A = B$ Type), ($M\colon A$), and ($M = N\colon A$). We write $\mathscr{J}[\sigma]$ for the obvious application of the explicit substitution σ, for example $(M = N\colon A)[\sigma]$ means $M[\sigma] = N[\sigma]\colon A[\sigma]$. For the explicit substitution calculus we do not need a notation for contexts with holes.

6.4.1 Rules of Inference

In the rules we frequently make use of substitutions $!_\Gamma\colon \Gamma \to \diamond$. In all rules where $!_\Gamma$ is used, the premise $!_\Gamma\colon \Gamma \to \diamond$ is implicit. Because of the rule (STR-$\diamond$EQ), it is clear that, for each context Γ, there is a unique substitution of that type. Furthermore, for contexts Γ and Δ, use the abbreviations $\dot{\pi}_1^{\Gamma,\Delta} = \mathrm{unit}^{-1} \circ (id * !_\Delta)$ and $\dot{\pi}_2^{\Gamma,\Delta} = \mathrm{unit}^{-1} \circ (id * !_\Gamma) \circ \mathrm{swap}$, which correspond to the projection maps $\Gamma * \Delta \to \Gamma$ and $\Gamma * \Delta \to \Delta$. Again we assume that any rule that mentions $\dot{\pi}_1^{\Gamma,\Delta}$ or $\dot{\pi}_2^{\Gamma,\Delta}$ has an implicit premise $\dot{\pi}_1^{\Gamma,\Delta}\colon \Gamma * \Delta \to \Gamma$ or $\dot{\pi}_2^{\Gamma,\Delta}\colon \Gamma * \Delta \to \Delta$. When clear from the context, we omit the context annotations, writing just $\dot{\pi}_1$ and $\dot{\pi}_2$.

Since we make the convention that no context may declare a variable more than once, we implicitly assume side-conditions ensuring that all the contexts mentioned in the rules have this property.

6.4.1.1 Bunches

$$\text{(BU-EMPTY)}\ \frac{}{\vdash \diamond \Downarrow \diamond\ \text{Bunch}} \qquad \text{(BU-ADD)}\ \frac{\vdash \Gamma \Downarrow \Gamma'\ \text{Bunch} \qquad \Gamma \vdash A \Downarrow A'\ \text{Type}}{\vdash \Gamma, x\colon A \Downarrow \Gamma', x\colon A'\ \text{Bunch}}$$

$$\text{(BU-MULT)}\ \frac{\vdash \Gamma \Downarrow \Gamma'\ \text{Bunch} \qquad \vdash \Delta \Downarrow \Delta'\ \text{Bunch}}{\vdash \Gamma * \Delta \Downarrow \Gamma' * \Delta'\ \text{Bunch}}$$

6.4.1.2 Projection and Substitution

$$\text{(PROJ)}\ \frac{\Gamma \vdash A\ \text{Type}}{\Gamma, x\colon A \vdash 1 \Downarrow x : A[\mathrm{w}]} \qquad \text{(SUBST)}\ \frac{\Gamma \vdash \mathscr{J} \Downarrow K \qquad \sigma \Downarrow \chi\colon \Delta \to \Gamma}{\Delta \vdash \mathscr{J}[\sigma] \Downarrow K\ \chi}$$

6.4.1.3 Structural maps

$$\text{(STR-ID)}\ \frac{\vdash \Gamma\ \text{Bunch}}{id\colon \Gamma \Rightarrow \Gamma} \qquad \text{(STR-COMP)}\ \frac{\sigma\colon \Gamma \Rightarrow \Delta \qquad \tau\colon \Delta \Rightarrow \Phi}{\tau \circ \sigma\colon \Gamma \Rightarrow \Phi}$$

$$\text{(STR-}*\text{)}\ \frac{\sigma\colon \Gamma \Rightarrow \Delta \qquad \tau\colon \Phi \Rightarrow \Psi}{\sigma * \tau\colon \Gamma * \Phi \Rightarrow \Delta * \Psi}\ \emptyset = (\Gamma \cup \Delta) \cap (\Phi \cup \Psi)$$

$$\text{(STR-LIFT)}\ \frac{\sigma\colon \Gamma \Rightarrow \Delta \qquad \Delta \vdash A\ \text{Type}}{\sigma_\bullet\colon (\Gamma, x\colon A[\sigma]) \Rightarrow (\Delta, x\colon A)}$$

$$\text{(STR-WEAK)}\ \frac{\Gamma \vdash A\ \text{Type}}{\mathrm{w}\colon (\Gamma, x\colon A) \Rightarrow \Gamma}$$

$$(\textsc{Str-Unit})\ \frac{\vdash \Gamma\ \mathrm{Bunch}}{\mathrm{unit}\colon \Gamma \Rightarrow \Gamma * \diamond} \qquad (\textsc{Str-Unit}^{-1})\ \frac{\vdash \Gamma\ \mathrm{Bunch}}{\mathrm{unit}^{-1}\colon \Gamma * \diamond \Rightarrow \Gamma}$$

$$(\textsc{Str-Swap})\ \frac{\vdash \Gamma\ \mathrm{Bunch} \qquad \vdash \Delta\ \mathrm{Bunch}}{\mathrm{swap}\colon \Gamma * \Delta \Rightarrow \Delta * \Gamma}$$

$$(\textsc{Str-Assoc})\ \frac{\vdash \Gamma\ \mathrm{Bunch} \qquad \vdash \Delta\ \mathrm{Bunch} \qquad \vdash \Phi\ \mathrm{Bunch}}{\mathrm{assoc}\colon (\Gamma * \Delta) * \Phi \Rightarrow \Gamma * (\Delta * \Phi)}$$

$$(\textsc{Str-Assoc}^{-1})\ \frac{\vdash \Gamma\ \mathrm{Bunch} \qquad \vdash \Delta\ \mathrm{Bunch} \qquad \vdash \Phi\ \mathrm{Bunch}}{\mathrm{assoc}^{-1}\colon \Gamma * (\Delta * \Phi) \Rightarrow (\Gamma * \Delta) * \Phi}$$

The reader may want to compare these rules for structural maps to those for $\Gamma \succcurlyeq \Delta$ in Section 4.4.

6.4.1.4 Substitutions

$$(\textsc{Sub-Str})\ \frac{\tau\colon \Gamma \Rightarrow \Delta}{\tau \Downarrow ()\colon \Gamma \to \Delta} \qquad (\textsc{Sub-Comp})\ \frac{\sigma \Downarrow \chi\colon \Gamma \to \Delta \qquad \tau \Downarrow \xi\colon \Delta \to \Phi}{\tau \circ \sigma \Downarrow \{(\xi \circ \chi)/\Phi\}\colon \Gamma \to \Phi}$$

$$(\textsc{Sub-}*)\ \frac{\sigma \Downarrow \chi\colon \Gamma \to \Delta \qquad \tau \Downarrow \xi\colon \Phi \to \Psi}{\sigma * \tau \Downarrow \chi\{\xi/\Psi\}\colon \Gamma * \Phi \to \Delta * \Psi}\ \emptyset = (\Gamma \cup \Delta) \cap (\Phi \cup \Psi)$$

$$(\textsc{Sub-Lift})\ \frac{\sigma \Downarrow \chi\colon \Gamma \to \Delta \qquad \Delta \vdash A\ \mathrm{Type}}{\sigma_\bullet \Downarrow \chi\colon (\Gamma, x\colon A[\sigma]) \to (\Delta, x\colon A)}$$

$$(\textsc{Sub-Sub})\ \frac{\Gamma \vdash M \Downarrow M' : A}{\langle M \rangle \Downarrow \{M'/x\}\colon \Gamma \to (\Gamma, x\colon A)}$$

6.4.1.5 Types

$$(\text{C-Ty})\ \frac{\sigma \Downarrow \chi\colon \Gamma \to \Delta}{\Gamma \vdash T(\sigma) \Downarrow T(x_1, \ldots, x_n)\ \chi\ \mathrm{Type}}\ T \in \mathscr{T}(\Delta),\ v(\Delta) = (x_1, \ldots, x_n)$$

$$(*\text{-Ty})\ \frac{\vdash A \Downarrow A'\ \mathrm{Type} \qquad \vdash B \Downarrow B'\ \mathrm{Type} \qquad \vdash \Gamma\ \mathrm{Bunch}}{\Gamma \vdash A * B \Downarrow A' * B'\ \mathrm{Type}}$$

$$(\Pi\text{-Ty})\ \frac{\Gamma \vdash A \Downarrow A'\ \mathrm{Type} \qquad \Gamma, x\colon A \vdash B \Downarrow B'\ \mathrm{Type}}{\Gamma \vdash \Pi A.B \Downarrow \Pi x\colon A'.B'\ \mathrm{Type}}$$

$$(\Pi^*\text{-Ty})\ \frac{\vdash A \Downarrow A'\ \mathrm{Type} \qquad \Gamma * x\colon A \vdash B \Downarrow B'\ \mathrm{Type}}{\Gamma \vdash \Pi^* A.B \Downarrow \Pi^* x\colon A'.B'\ \mathrm{Type}}$$

6.4.1.6 Terms

$$(\text{C-Tm})\ \frac{\Delta \vdash A\ \mathrm{Type} \qquad \sigma \Downarrow \chi\colon \Gamma \to \Delta}{\Gamma \vdash f(\sigma) \Downarrow f(x_1, \ldots, x_n)\ \chi : A[\sigma]}\ f \in \mathscr{T}(\Delta; A),\ v(\Delta) = (x_1, \ldots, x_n)$$

$$(*\text{-I})\ \frac{\vdash A \Downarrow A' \text{ Type} \qquad \vdash B \Downarrow B' \text{ Type} \qquad \Gamma \vdash M \Downarrow M' : A[!_\Gamma] \qquad \Delta \vdash N \Downarrow N' : B[!_\Delta]}{\Gamma * \Delta \vdash \text{pair}^*_{A,B}(M,N) \Downarrow \text{pair}^*_{A',B'}(M',N') : A*B}$$

$$(*\text{-E})\ \frac{\vdash A \Downarrow A' \text{ Type} \qquad \vdash B \Downarrow B' \text{ Type} \qquad \Gamma \vdash M \Downarrow M' : A*B}{\langle M/A*B\rangle \Downarrow \{r_{A',B'}(M')/y\}\{l_{A',B'}(M')/x\} : \Gamma \to (x\colon A) * (y\colon B)}$$

$$(\Pi\text{-I})\ \frac{\Gamma \vdash A \Downarrow A' \text{ Type} \qquad \Gamma, x\colon A \vdash M \Downarrow M' : B}{\Gamma \vdash \lambda A.M \Downarrow \lambda x\colon A'.M' : \Pi A.B}$$

$$(\Pi\text{-E})\ \frac{\Gamma \vdash A \Downarrow A' \text{ Type} \qquad \Gamma, x\colon A \vdash B \Downarrow B' \text{ Type} \qquad \Gamma \vdash M \Downarrow M' : \Pi A.B \qquad \Gamma \vdash N \Downarrow N' : A}{\Gamma \vdash \text{app}_{[A]B}(M,N) \Downarrow \text{app}_{(x:A')B'}(M',N') : B[\langle N\rangle]}$$

$$(\Pi^*\text{-I})\ \frac{\vdash A \Downarrow A' \text{ Type} \qquad \Gamma * x\colon A \vdash M \Downarrow M' : B}{\Gamma \vdash \lambda^* A.M \Downarrow \lambda^* x\colon A'.M' : \Pi^* A.B}$$

$$(\Pi^*\text{-E})\ \frac{\begin{array}{c}\vdash A \Downarrow A' \text{ Type} \qquad \Gamma \vdash M \Downarrow M' : \Pi^* A.B \\ \Gamma * x\colon A \vdash B \Downarrow B' \text{ Type} \qquad \Delta \vdash N \Downarrow N' : A[!_\Delta]\end{array}}{\Gamma * \Delta \vdash \text{app}^*_{[A]B}(M,N) \Downarrow \text{app}^*_{(x:A')B'}(M',N') : B[id * ((!_\Delta)_\bullet \circ \langle N\rangle)]}$$

6.4.1.7 Conversions

$$(\text{Bu-Conv})\ \frac{\Gamma \vdash \mathscr{J} \qquad \vdash \Gamma = \Delta \text{ Bunch}}{\Delta \vdash \mathscr{J}} \qquad (\text{Ty-Conv})\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash A = B \text{ Type}}{\Gamma \vdash M : B}$$

$$(\text{Ty-Eq-Conv})\ \frac{\Gamma \vdash M = N : A \qquad \Gamma \vdash A = B \text{ Type}}{\Gamma \vdash M = N : B}$$

$$(\Rightarrow\text{-Bu-Conv})\ \frac{\vdash \Phi = \Gamma \text{ Bunch} \qquad \sigma\colon \Gamma \Rightarrow \Delta \qquad \vdash \Delta = \Psi \text{ Bunch}}{\sigma\colon \Phi \Rightarrow \Psi}$$

$$(\to\text{-Bu-Conv})\ \frac{\vdash \Phi = \Gamma \text{ Bunch} \qquad \sigma \Downarrow \chi\colon \Gamma \to \Delta \qquad \vdash \Delta = \Psi \text{ Bunch}}{\sigma \Downarrow \chi\colon \Phi \to \Psi}$$

$$(\to\text{-Eq-Bu-Conv})\ \frac{\vdash \Phi = \Gamma \text{ Bunch} \qquad \sigma = \tau\colon \Gamma \to \Delta \qquad \vdash \Delta = \Psi \text{ Bunch}}{\sigma = \tau\colon \Phi \to \Psi}$$

6.4.1.8 General equations

$$\frac{\Gamma \vdash \mathscr{J}}{\Gamma \vdash \mathscr{J}[id] = \mathscr{J}} \qquad \frac{\tau\colon \Gamma \to \Delta \qquad \sigma\colon \Delta \to \Phi \qquad \Phi \vdash \mathscr{J}}{\Gamma \vdash \mathscr{J}[\sigma][\tau] = \mathscr{J}[\sigma \circ \tau]} \qquad \frac{\sigma = \tau\colon \Gamma \to \Delta \qquad \Delta \vdash \mathscr{J}}{\Gamma \vdash \mathscr{J}[\sigma] = \mathscr{J}[\tau]}$$

6.4.1.9 Equations for Bunches

(reflexivity, symmetry, transitivity)

$$(\text{Bu-AEq})\ \frac{\vdash \Gamma = \Delta \text{ Bunch} \qquad \Gamma \vdash A = B \text{ Type}}{\vdash (\Gamma, x\colon A) = (\Delta, x\colon B) \text{ Bunch}}$$

$$(\text{Bu-MEq})\ \frac{\vdash \Gamma = \Delta \text{ Bunch} \qquad \vdash \Phi = \Psi \text{ Bunch}}{\vdash \Gamma * \Phi = \Delta * \Psi \text{ Bunch}}$$

6.4.1.10 Equations for Substitutions

(reflexivity, symmetry, transitivity)

$$(\textsc{Sub-}\diamond\textsc{Eq})\ \frac{\sigma\colon \Gamma \to \diamond \qquad \tau\colon \Gamma \to \diamond}{\sigma = \tau\colon \Gamma \to \diamond}$$

$$(\textsc{Sub-AEq})\ \frac{\sigma,\tau\colon \Gamma \to (\Delta, x\colon A) \qquad \mathrm{w}\circ\sigma = \mathrm{w}\circ\tau\colon \Gamma \to \Delta \qquad \Gamma \vdash 1[\sigma] = 1[\tau] : A[\mathrm{w}][\sigma] = A[\mathrm{w}][\tau]}{\sigma = \tau\colon \Gamma \to (\Delta, x\colon A)}$$

$$(\textsc{Sub-MEq})\ \frac{\sigma,\tau\colon \Gamma \to (\Delta_1 * \Delta_2) \qquad \dot{\pi}_1\circ\sigma = \dot{\pi}_1\circ\tau\colon \Gamma \to \Delta_1 \qquad \dot{\pi}_2\circ\sigma = \dot{\pi}_2\circ\tau\colon \Gamma \to \Delta_2}{\sigma = \tau\colon \Gamma \to (\Delta_1 * \Delta_2)}$$

$$(\textsc{Cat-Id})\ \frac{\sigma\colon \Gamma \to \Delta}{id\circ\sigma = \sigma\circ id = \sigma\colon \Gamma \to \Delta}$$

$$(\textsc{Cat-Cmp})\ \frac{\sigma\colon \Gamma \to \Delta \qquad \tau\colon \Delta \to \Phi \qquad \rho\colon \Phi \to \Psi}{(\rho\circ\tau)\circ\sigma = \rho\circ(\tau\circ\sigma)\colon \Gamma \to \Psi}$$

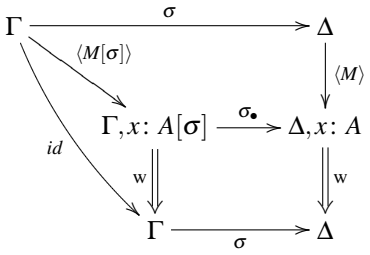
$$(*\textsc{-Id})\ \frac{\vdash \Gamma\ \mathrm{Bunch} \qquad \vdash \Delta\ \mathrm{Bunch}}{id * id = id\colon \Gamma * \Delta \to \Gamma * \Delta}$$

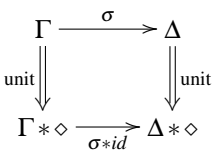
$$(*\textsc{-Cmp})\ \frac{\sigma\colon \Gamma \to \Delta \qquad \sigma'\colon \Gamma' \to \Delta' \qquad \tau\colon \Delta \to \Phi \qquad \tau'\colon \Delta' \to \Phi'}{(\tau\circ\sigma) * (\tau'\circ\sigma') = (\tau * \tau')\circ(\sigma * \sigma')\colon \Gamma * \Gamma' \to \Phi * \Phi'}\ \emptyset = \Delta \cap \Delta'$$

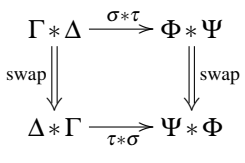
$$((-)_\bullet\textsc{-Id})\ \frac{\Gamma \vdash A\ \mathrm{Type}}{id_\bullet = id\colon \Gamma, x\colon A \to \Gamma, x\colon A}$$

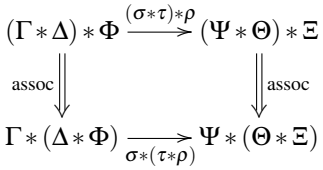
$$((-)_\bullet\textsc{-Cmp})\ \frac{\sigma\colon \Gamma \to \Delta \qquad \tau\colon \Delta \to \Phi \qquad \Phi \vdash A\ \mathrm{Type}}{(\tau\circ\sigma)_\bullet = \tau_\bullet\circ\sigma_\bullet\colon \Gamma, x\colon A[\tau][\sigma] \to \Phi, x\colon A}$$

We state the rest of the equations for substitutions in diagram form. In addition to the usual definition of commutation that any two paths with the same start and end are equal, we also presuppose that all the morphisms in the diagrams are derivable. For example, the next diagram presupposes $\sigma\colon \Gamma \to \Delta$, $\Delta \vdash M : A$ and $x \notin \Gamma, \Delta$.









In addition, we assume the commuting diagrams from Definitions 2.5.1 and 2.5.2 stating that $*$ is a symmetric monoidal structure. Finally, we also assume equations stating that unit^{-1} and assoc^{-1} are inverses of unit and assoc respectively.

Because of the rule (SUB-STR), we can speak of equality also for structural maps. When referring to equality involving a structural map $\Gamma \Rightarrow \Delta$, we mean the equality of type $\Gamma \to \Delta$ that arises by implicitly inserting (SUB-STR).

6.4.1.11 Equations for Types

(reflexivity, symmetry, transitivity)

Substitution on types

$$\text{(C-SUB)}\ \frac{\Delta \vdash T(\sigma)\ \text{Type} \qquad \tau\colon \Gamma \to \Delta}{\Gamma \vdash T(\sigma)[\tau] = T(\sigma \circ \tau)\ \text{Type}} \qquad \text{(}*\text{-SUB)}\ \frac{\Delta \vdash A*B\ \text{Type} \qquad \sigma\colon \Gamma \to \Delta}{\Gamma \vdash (A*B)[\sigma] = A*B\ \text{Type}}$$

$$\text{(}\Pi\text{-SUB)}\ \frac{\Delta \vdash \Pi A.B\ \text{Type} \qquad \sigma\colon \Gamma \to \Delta}{\Gamma \vdash (\Pi A.B)[\sigma] = (\Pi A[\sigma].B[\sigma_\bullet])\ \text{Type}}$$

$$\text{(}\Pi^*\text{-SUB)}\ \frac{\Delta \vdash \Pi^* A.B\ \text{Type} \qquad \sigma\colon \Gamma \to \Delta}{\Gamma \vdash (\Pi^* A.B)[\sigma] = (\Pi^* A.B[\sigma * id])\ \text{Type}}$$

Type congruences

$$\text{(C-TY-CGR)}\ \frac{\sigma = \tau\colon \Gamma \to \Delta}{\Gamma \vdash T(\sigma) = T(\tau)\ \text{Type}}\ T \in \mathscr{T}(\Delta)$$

$$\text{(}*\text{-TY-CGR)}\ \frac{\vdash A_1 = A_2\ \text{Type} \qquad \vdash B_1 = B_2\ \text{Type}}{\Gamma \vdash A_1 * B_1 = A_2 * B_2\ \text{Type}}$$

$$\text{(}\Pi\text{-TY-CGR)}\ \frac{\Gamma \vdash A_1 = A_2\ \text{Type} \qquad \Gamma, x\colon A_1 \vdash B_1 = B_2\ \text{Type}}{\Gamma \vdash \Pi A_1.B_1 = \Pi A_2.B_2\ \text{Type}}$$

$$\text{(}\Pi^*\text{-TY-CGR)}\ \frac{\vdash A_1 = A_2\ \text{Type} \qquad \Gamma * x\colon A_1 \vdash B_1 = B_2\ \text{Type}}{\Gamma \vdash \Pi^* A_1.B_1 = \Pi^* A_2.B_2\ \text{Type}}$$

Type axioms

$$\text{(TY-AXIOM)}\ \frac{\Gamma \vdash A\ \text{Type} \qquad \Gamma \vdash B\ \text{Type}}{\Gamma \vdash A = B\ \text{Type}}\ A = B \in \mathscr{T}^{\mathscr{E}}(\Gamma)$$

6.4.1.12 Equations for Terms

(reflexivity, symmetry, transitivity)

Substitution on terms

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash 1[\langle M \rangle] = M : A}$$

$$\frac{\sigma : \Gamma \to \Delta \qquad \Delta \vdash A \text{ Type}}{\Gamma, x : A[\sigma] \vdash 1[\sigma_\bullet] = 1 : A[\mathrm{w}][\sigma_\bullet]\ (= A[\sigma][\mathrm{w}])}$$

$$\frac{\Delta \vdash f(\sigma) : A \qquad \tau : \Gamma \to \Delta}{\Gamma \vdash f(\sigma)[\tau] = f(\sigma \circ \tau) : A[\tau]}$$

$$\frac{\begin{array}{ccc} \vdash A \text{ Type} & \Gamma \vdash M : A[!_\Gamma] & \sigma : \Gamma' \to \Gamma \\ \vdash B \text{ Type} & \Delta \vdash N : B[!_\Delta] & \tau : \Delta' \to \Delta \end{array}}{\Gamma' * \Delta' \vdash (\mathrm{pair}^*_{A,B}(M,N))[\sigma * \tau] = \mathrm{pair}^*_{A,B}(M[\sigma], N[\tau]) : A * B}\ \emptyset = \Gamma \cap \Delta$$

$$\frac{\vdash A \text{ Type} \qquad \vdash B \text{ Type} \qquad \Delta \vdash M : A * B \qquad \sigma : \Gamma \to \Delta}{\langle M/A * B \rangle \circ \sigma = \langle M[\sigma]/A * B \rangle : \Gamma \to (x : A) * (y : B)}$$

$$\frac{\Delta, x : A \vdash M : B \qquad \sigma : \Gamma \to \Delta}{\Gamma \vdash (\lambda A.M)[\sigma] = \lambda A[\sigma].M[\sigma_\bullet] : (\Pi A.B)[\sigma]}$$

$$\frac{\Delta, x : A \vdash B \text{ Type} \qquad \Delta \vdash M : \Pi A.B \qquad \Delta \vdash N : A \qquad \sigma : \Gamma \to \Delta}{\Gamma \vdash (\mathrm{app}_{[A]B}(M,N))[\sigma] = \mathrm{app}_{[A[\sigma]]B[\sigma_\bullet]}(M[\sigma], N[\sigma]) : B[\langle N \rangle][\sigma]}$$

$$\frac{\Delta * x : A \vdash M : B \qquad \sigma : \Gamma \to \Delta}{\Gamma \vdash (\lambda^* A.M)[\sigma] = \lambda^* A.M[\sigma * id] : (\Pi^* x : A.B)[\sigma]}$$

$$\frac{\Gamma * x : A \vdash B \text{ Type} \qquad \Gamma \vdash M : \Pi^* A.B \qquad \Delta \vdash N : A \qquad \sigma : \Phi \to \Gamma \qquad \tau : \Psi \to \Delta}{\Phi * \Psi \vdash (\mathrm{app}^*_{[A]B}(M,N))[\sigma * \tau] = \mathrm{app}^*_{[A]B[\sigma * id]}(M[\sigma], N[\tau]) : B[\sigma * ((!_\Psi)_\bullet \circ \langle N[\tau] \rangle)]}\ \emptyset = \Gamma \cap \Delta$$

Term axioms

$$(\textsc{Axiom})\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash M = N : A}\ M = N \in \mathscr{T}^{\mathscr{E}}(\Gamma; A)$$

Equations for $*$-terms

$$(*\text{-I-}\textsc{Cgr})\ \frac{\begin{array}{cc} \vdash A = A' \text{ Type} & \Gamma \vdash M = M' : A[!_\Gamma] \\ \vdash B = B' \text{ Type} & \Delta \vdash N = N' : B[!_\Delta] \end{array}}{\Gamma * \Delta \vdash \mathrm{pair}^*_{A,B}(M,N) = \mathrm{pair}^*_{A',B'}(M',N') : A * B}$$

$$(*\text{-E-}\textsc{Cgr})\ \frac{\vdash A = A' \text{ Type} \qquad \vdash B = B' \text{ Type} \qquad \Gamma \vdash M = M' : A * B}{\langle M/A * B \rangle = \langle M'/A' * B' \rangle : \Gamma \to (x : A) * (y : B)}$$

$$(*\text{-}\beta)\ \frac{\vdash A\ \mathrm{Type} \qquad \vdash B\ \mathrm{Type} \qquad \Gamma \vdash M : A \qquad \Delta \vdash N : B}{\langle \mathrm{pair}^*_{A,B}(M,N)/A*B\rangle = ((!_\Gamma)_\bullet \circ \langle M\rangle) * ((!_\Delta)_\bullet \circ \langle N\rangle) \colon \Gamma * \Delta \to (x\colon A) * (y\colon B)}$$

$$(*\text{-}\eta)\ \frac{\vdash A\ \mathrm{Type} \qquad \vdash B\ \mathrm{Type} \qquad \Gamma \vdash M : A*B}{\Gamma \vdash (1*1)[\langle M/A*B\rangle] = M : A*B}$$

$$(\textsc{Inject})\ \frac{\begin{array}{lll} \vdash A\ \mathrm{Type} & \Gamma \vdash M : A*B & \Gamma \vdash 1[\dot\pi_1 \circ \langle M/A*B\rangle] = 1[\dot\pi_1 \circ \langle N/A*B\rangle] : A[!_\Gamma] \\ \vdash B\ \mathrm{Type} & \Gamma \vdash N : A*B & \Gamma \vdash 1[\dot\pi_2 \circ \langle M/A*B\rangle] = 1[\dot\pi_2 \circ \langle N/A*B\rangle] : B[!_\Gamma] \end{array}}{\Gamma \vdash M = N : A*B}$$

Equations for Π-terms

$$(\Pi\text{-I-CGR})\ \frac{\Gamma \vdash A_1 = A_2\ \mathrm{Type} \qquad \Gamma, x\colon A_1 \vdash M_1 = M_2 : B}{\Gamma \vdash \lambda A_1.M_1 = \lambda A_2.M_2 : \Pi x\colon A_1.B}$$

$$(\Pi\text{-E-CGR})\ \frac{\begin{array}{ll} \Gamma \vdash A_1 = A_2\ \mathrm{Type} & \Gamma \vdash M_1 = M_2 : \Pi A_1.B_1 \\ \Gamma, x\colon A_1 \vdash B_1 = B_2\ \mathrm{Type} & \Gamma \vdash N_1 = N_2 : A_1 \end{array}}{\Gamma \vdash \mathrm{app}_{[A_1]B_1}(M_1,N_1) = \mathrm{app}_{[A_2]B_2}(M_2,N_2) : B_1[\langle N_1\rangle]}$$

$$(\Pi\text{-}\beta)\ \frac{\Gamma, x\colon A \vdash M : B \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathrm{app}_{[A]B}(\lambda A.M,N) = M[\langle N\rangle] : B[\langle N\rangle]}$$

$$(\Pi\text{-}\eta)\ \frac{\Gamma, x\colon A \vdash B\ \mathrm{Type} \qquad \Gamma \vdash M : \Pi A.B}{\Gamma \vdash \lambda A.(\mathrm{app}_{[A]B}(M,1)) = M : \Pi A.B}$$

Equations for Π^*-terms

$$(\Pi^*\text{-I-CGR})\ \frac{\vdash A_1 = A_2\ \mathrm{Type} \qquad \Gamma * x\colon A_1 \vdash M_1 = M_2 : B}{\Gamma \vdash \lambda^* A_1.M_1 = \lambda^* A_2.M_2 : \Pi^* A_1.B}$$

$$(\Pi^*\text{-E-CGR})\ \frac{\begin{array}{rl} & \vdash A_1 = A_2\ \mathrm{Type} \\ \Gamma_1 \vdash M_1 : \Pi^* A_1.B_1 & \Phi \vdash B_1[\sigma] = B_2[\tau]\ \mathrm{Type} \\ \Gamma_2 \vdash M_2 : \Pi^* A_2.B_2 & \Phi \vdash (\Pi^* A_1.B_1)[\dot\pi_1 \circ \sigma] = (\Pi^* A_2.B_2)[\dot\pi_1 \circ \tau]\ \mathrm{Type} \\ \sigma\colon \Phi \to \Gamma_1 * x\colon A_1 & \Phi \vdash M_1[\dot\pi_1 \circ \sigma] = M_2[\dot\pi_1 \circ \tau] : (\Pi^* A_1.B_1)[\dot\pi_1 \circ \sigma] \\ \tau\colon \Phi \to \Gamma_2 * x\colon A_2 & \Phi \vdash 1[\dot\pi_2 \circ \sigma] = 1[\dot\pi_2 \circ \tau] : A_1[!_\Phi] \end{array}}{\Phi \vdash (\mathrm{app}^*_{[A_1]B_1}(M_1,1))[\sigma] = (\mathrm{app}^*_{[A_2]B_2}(M_2,1))[\tau] : B_1[\sigma]}$$

$$(\Pi^*\text{-}\beta)\ \frac{\Gamma * x\colon A \vdash M : B \qquad \Delta \vdash N : A}{\Gamma * \Delta \vdash \mathrm{app}^*_{[A]B}((\lambda^* A.M),N) = M[id * ((!_\Delta)_\bullet \circ \langle N\rangle)] : B[id * ((!_\Delta)_\bullet \circ \langle N\rangle)]}$$

$$(\Pi^*\text{-}\eta)\ \frac{\Gamma * x\colon A \vdash B\ \mathrm{Type} \qquad \Gamma \vdash M : \Pi^* A.B}{\Gamma \vdash \lambda^* A.(\mathrm{app}^*_{[A]B}(M,1)) = M : \Pi^* A.B}$$

6.4.2 Basic Properties

We state some basic properties of **ES**, which are proved in the same way as in Chapter 4.

Lemma 6.4.1. *If* $\Gamma \vdash \Pi A.B \Downarrow \Pi x\colon A'.B'$ Type *then* $\Gamma \vdash A \Downarrow A'$ Type *and* $\Gamma, x\colon A \vdash B \Downarrow B'$ Type.

Lemma 6.4.2. *If* $\Gamma \vdash \Pi^* A.B \Downarrow \Pi^* x\colon A'.B'$ Type *then* $\vdash A \Downarrow A'$ Type *and* $\Gamma * x\colon A \vdash B \Downarrow B'$ Type.

Lemma 6.4.3 (Validity)**.**

1. *If* $\Gamma \vdash \mathscr{J}$ *then* $\vdash \Gamma$ Bunch.
2. *If* $\vdash \Gamma = \Delta$ Bunch *then* $\vdash \Gamma$ Bunch *and* $\vdash \Delta$ Bunch.
3. *If* $\Gamma \vdash A = B$ Type *then* $\Gamma \vdash A$ Type *and* $\Gamma \vdash B$ Type.
4. *If* $\Gamma \vdash M : A$ *then* $\Gamma \vdash A$ Type.
5. *If* $\Gamma \vdash M = N : A$ *then* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : A$.
6. *If* $\sigma\colon \Gamma \to \Delta$ *then* $\vdash \Gamma$ Bunch *and* $\vdash \Delta$ Bunch.
7. *If* $\sigma = \tau\colon \Gamma \to \Delta$ *then* $\sigma\colon \Gamma \to \Delta$ *and* $\tau\colon \Gamma \to \Delta$.
8. *If* $\sigma\colon \Gamma \Rightarrow \Delta$ *then* $\vdash \Gamma$ Bunch *and* $\vdash \Delta$ Bunch.

Lemma 6.4.4.

1. *If* $id\colon \Gamma \to \Delta$ *then* $\vdash \Gamma = \Delta$ Bunch
2. *If* $\tau \circ \sigma\colon \Gamma \to \Delta$ *then there exists a context* Φ *such that* $\sigma\colon \Gamma \to \Phi$ *and* $\tau\colon \Phi \to \Delta$.
3. *If* $\sigma_\bullet\colon (\Gamma, x\colon B) \to (\Delta, x\colon A)$ *then* $\sigma\colon \Gamma \to \Delta$ *and* $\Gamma \vdash B = A[\sigma]$ Type.
4. *If* $\langle M/A * B\rangle \Downarrow \{r_{A'',B''}(M'')/y\}\{l_{A',B'}(M')/x\}\colon \Gamma \to \Delta$ *then* $A' = A''$, $B' = B''$ *and* $M' = M''$, *as well as* $\vdash A \Downarrow A'$ Type *and* $\vdash B \Downarrow B'$ Type *and* $\Gamma \vdash M \Downarrow M' : A*B$ *and* $\vdash \Delta = (x\colon A) * (y\colon B)$ Bunch.

Proof. We show the last case. The other cases follow by similar examination of the derivations.

Any derivation of $\langle M/A * B\rangle \Downarrow \{r_{A'',B''}(M'')/y\}\{l_{A',B'}(M')/x\}\colon \Gamma \to \Delta$ ends in ($*$-E) followed by zero or more applications of ($\to$-BU-CONV). By reflexivity and transitivity of bunch equality we may assume that it ends in ($*$-E) followed by exactly one application of ($\to$-BU-CONV). By examining these rules we get $A' = A''$, $B' = B''$ and $M' = M''$, as well as derivations of $\vdash A \Downarrow A'$ Type and $\vdash B \Downarrow B'$ Type and $\vdash \Gamma = \Gamma'$ Bunch and $\vdash (x\colon A) * (y\colon B) = \Delta$ Bunch and $\Gamma' \vdash M \Downarrow M' : A*B \Downarrow A'*B'$, thus completing this case. □

6.4.3 Properties of the Syntax-Translation

For the following lemma, we extend substitution with a bijective renaming α in the evident way to contexts Γ in **ES**.

Lemma 6.4.5 (Closure under bijective renaming)**.** *For any bijective renaming of variables α, the following hold.*

- *If* $\vdash \Gamma$ Bunch *is derivable then so is* $\vdash (\Gamma\ \alpha)$ Bunch.
- *If* $\Gamma \vdash J \Downarrow K$ *is derivable then so is* $(\Gamma\ \alpha) \vdash J \Downarrow (K\ \alpha)$.
- *If* $\sigma \Downarrow \chi : \Gamma \to \Delta$ *is derivable then so is* $\sigma \Downarrow (\alpha^{-1} \circ \chi \circ \alpha) : \Gamma\ \alpha \to \Delta\ \alpha$.

Moreover, the resulting derivations have the same shape and size as the original derivations.

Proof. By induction on derivations. We consider representative cases.

- For the rule (Π-TY), the induction hypothesis gives us $(\Gamma\ \alpha) \vdash A \Downarrow A'\ \alpha$ Type and $(\Gamma\ \alpha), \alpha(x) : A \vdash B \Downarrow B'\ \alpha$ Type. For a bijective renaming α, we have $(\Pi x : A.B)\ \alpha = \Pi\alpha(x) : (A\ \alpha).(B\ \alpha)$ by Lemma 6.3.1. Using (Π-TY) we can therefore derive the required $(\Gamma\ \alpha) \vdash (\Pi x : A.B)\ \alpha$ Type.
- For the rule (SUBST), we have $\Gamma\ \alpha \vdash J \Downarrow K\ \alpha$ and $\sigma \Downarrow (\alpha^{-1} \circ \chi \circ \alpha) : (\Delta\ \alpha) \to (\Gamma\ \alpha)$ by induction hypothesis. Using (SUBST) we can therefore derive $\Delta\ \alpha \vdash J \Downarrow K\ \alpha\ (\alpha^{-1} \circ \chi \circ \alpha)$ which, by the properties of syntactic substitution, is the same as the required $\Delta\ \alpha \vdash J \Downarrow K\ \chi\ \alpha$.
- For the rule (SUB-COMP), the induction hypothesis gives $\sigma \Downarrow (\alpha^{-1} \circ \chi \circ \alpha) : (\Gamma\ \alpha) \to (\Delta\ \alpha)$ and $\tau \Downarrow (\alpha^{-1} \circ \xi \circ \alpha) : (\Delta\ \alpha) \to (\Phi\ \alpha)$. The required sequent follows using the rule (SUB-COMP) and the (general) equality $(\alpha^{-1} \circ (\{\chi/\Psi\}) \circ \alpha) = \{(\alpha^{-1} \circ \chi \circ \alpha)/\alpha(\Psi)\}$.
- For the rule (SUB-$*$), we make use of the general equality $(\alpha^{-1} \circ \chi \circ \alpha)\{(\alpha^{-1} \circ \xi \circ \alpha)/\alpha(\Psi)\} = \alpha^{-1} \circ (\chi\{\xi/\Psi\}) \circ \alpha$.

□

The following two lemmas are shown by straightforward induction on derivations.

Lemma 6.4.6.

1. *If* $\Gamma \vdash A \Downarrow A'$ Type *then* $FV(A') \subseteq FV(\Gamma)$.
2. *If* $\Gamma \vdash M \Downarrow M' : A$ *then* $FV(M') \subseteq FV(\Gamma)$.
3. *If* $\sigma \Downarrow \chi : \Gamma \to \Delta$ *then* $FV(\chi(x)) \subseteq FV(\Gamma)$ *holds for all* $x \in FV(\Delta)$.
4. *If* $\sigma \Downarrow \chi : \Gamma \to \Delta$ *then* $\chi(x) = x$ *holds for all* $x \notin FV(\Delta)$.
5. *If* $\sigma : \Gamma \Rightarrow \Delta$ *then* $FV(\Delta) \subseteq FV(\Gamma)$.

Lemma 6.4.7.

1. *If* $\vdash \Gamma$ Bunch *then* $v(\Gamma)$ *is a list of pairwise distinct variables.*
2. *If* $\vdash \Gamma = \Delta$ Bunch *then* $v(\Gamma) = v(\Delta)$.
3. *If* $\sigma \colon \Gamma \Rightarrow \Delta$ *and* $\sigma \colon \Phi \Rightarrow \Psi$, *then* $v(\Gamma) = v(\Phi)$ *implies* $v(\Delta) = v(\Psi)$.

We now show that the syntactic translation in the rules for **ES** is unique, so that is makes sense to speak of *the* syntactic translation of a substitution/type/term/bunch.

Lemma 6.4.8.

1. *If* $\sigma \Downarrow \chi \colon \Gamma \to \Delta$ *and* $\sigma \Downarrow \chi' \colon \Gamma \to \Delta$ *then* $\chi = \chi'$.
2. *If* $\Gamma \vdash A \Downarrow A'$ Type *and* $\Gamma \vdash A \Downarrow A''$ Type *then* $A' = A''$.
3. *If* $\Gamma \vdash M \Downarrow M' : A$ *and* $\Gamma \vdash M \Downarrow M'' : A$ *then* $M' = M''$.

Proof. The proof goes by simultaneous induction on the derivations of the respective statements. In order to deal with conversion rules and composition of substitution, we strengthen the induction hypothesis to:

$1'$. If $\sigma \Downarrow \chi \colon \Gamma \to \Delta$ and $\sigma \Downarrow \chi' \colon \Phi \to \Psi$ with $v(\Gamma) = v(\Phi)$, then there exists a bijective renaming α satisfying $v(\Psi) = v(\Delta\ \alpha^{-1})$ and $\chi' =_\Psi \alpha \circ \chi$.

$2'$. If $\Gamma \vdash A \Downarrow A'$ Type and $\Delta \vdash A \Downarrow A''$ Type and $v(\Gamma) = v(\Delta)$ then $A' = A''$.

$3'$. If $\Gamma \vdash M \Downarrow M' : A$ and $\Delta \vdash M \Downarrow M'' : B$ and $v(\Gamma) = v(\Delta)$ then $M' = M''$.

We show these points by simultaneous induction on the size of the derivations. The induction is on the size because we may need to apply the induction hypothesis to renamed derivations.

$1'$. Suppose we have derivations of $\sigma \Downarrow \chi \colon \Gamma \to \Delta$ and $\sigma \Downarrow \chi' \colon \Phi \to \Psi$ with $v(\Gamma) = v(\Phi)$. If either derivation ends in ($\to$-BU-CONV), then, by Lemma 6.4.7, we can immediately use the induction hypothesis. Since the other rules are syntax-directed, we can therefore assume that both derivation end in an application of the same rule. Case distinction on the last rule.

- (SUB-STR). Both χ and χ' are $()$ and, by Lemma 6.4.7.3, $v(\Delta) = v(\Delta')$. The assertion follows with $\alpha = id$.
- (SUB-COMP). We have $\sigma_2 \circ \sigma_1 \Downarrow \chi_2 \circ \chi_1 \colon \Gamma \to \Theta \to \Delta$ and $\sigma_2 \circ \sigma_1 \Downarrow \chi_2' \circ \chi_1' \colon \Phi \to \Xi \to \Psi$. By induction hypothesis, we have α such that $v(\Xi) = v(\Theta\ \alpha^{-1})$ and $\alpha \circ \chi_1 =_\Xi \chi_1'$. Using Lemma 6.4.5, we get a derivation of $\sigma_2 \Downarrow \alpha^{-1} \circ \chi_2' \circ \alpha \colon (\Xi\ \alpha) \to (\Psi\ \alpha)$ to which we can apply the induction hypothesis to obtain a renaming β such that $v(\Psi\ \alpha) = v(\Delta\ \beta^{-1})$ and $\beta \circ \chi_2 =_\Psi \alpha^{-1} \circ \chi_2' \circ \alpha$. Then we have $\alpha \circ \beta \circ \chi_2 \circ \chi_1 =_\Psi \alpha \circ \alpha^{-1} \circ \chi_2' \circ \alpha \circ \chi_1 =_\Psi \chi_2' \circ \alpha \circ \chi_1 =_\Psi \chi_2' \circ \chi_1'$. Hence, we take $\alpha \circ \beta$ as the required renaming. Notice that $v(\Delta\ (\alpha \circ \beta)^{-1}) = v(\Delta\ (\alpha^{-1} \circ \beta^{-1})) = v(\Delta\ \alpha^{-1}\ \beta^{-1}) = v(\Psi)$ holds.

- (SUB-$*$). We have $\sigma_1 \Downarrow \chi_1 \colon \Gamma_1 \to \Delta_1$, $\sigma_2 \Downarrow \chi_2 \colon \Gamma_2 \to \Delta_2$, $\sigma_1 \Downarrow \chi_1' \colon \Phi_1 \to \Psi_1$ as well as $\sigma_2 \Downarrow \chi_2' \colon \Phi_2 \to \Psi_2$, where $\Gamma = \Gamma_1 * \Gamma_2$, $\Delta = \Delta_1 * \Delta_2$, $\Phi = \Phi_1 * \Phi_2$, $\Psi = \Psi_1 * \Psi_2$ and $\emptyset = (\Gamma_1 \cup \Delta_1) \cap (\Gamma_2 \cup \Delta_2) = (\Phi_1 \cup \Psi_1) \cap (\Phi_2 \cup \Psi_2)$. Moreover, $\chi = \chi_1\{\chi_2/\Delta_2\}$ and $\chi' = \chi_1'\{\chi_2'/\Psi_2\}$ hold. By induction hypothesis, we have α_1 and α_2 such that, for $i = 1, 2$, both $v(\Psi_i) = v(\Delta_i\ \alpha_i^{-1})$ and $\chi_i' =_{\Psi_i} \alpha_i \circ \chi_i$ hold. Since $\emptyset = \Psi_1 \cap \Psi_2 = \Delta_1 \cap \Delta_2$, the function α_0 defined by

  $$\alpha_0(x) = \begin{cases} \alpha_1(x) & \text{if } x \in \Psi_1 \\ \alpha_2(x) & \text{if } x \in \Psi_2 \end{cases}$$

  maps each element of $\Psi_1 \cup \Psi_2$ bijectively to an element of $\Delta_1 \cup \Delta_2$. This function α_0 can be extended to a bijection α on all variables. This gives $v(\Psi_1 * \Psi_2) = v((\Delta_1 * \Delta_2)\ \alpha^{-1})$. Furthermore, we have $(\alpha \circ \chi_i)(x) = (\alpha_i \circ \chi_i)(x)$ for any $x \in \Delta_i$, using which we obtain the required equation $\alpha \circ \chi =_{\Psi_1 * \Psi_2} \alpha \circ (\chi_1\{\chi_2/\Delta_2\}) =_{\Psi_1 * \Psi_2} (\alpha \circ \chi_1)\{\alpha \circ \chi_2/\alpha^{-1}(\Delta_2)\} =_{\Psi_1 * \Psi_2} (\alpha \circ \chi_1)\{\alpha \circ \chi_2/\Psi_2\} =_{\Psi_1 * \Psi_2} (\alpha_1 \circ \chi_1)\{\alpha_2 \circ \chi_2/\Psi_2\} =_{\Psi_1 * \Psi_2} \chi_1'\{\chi_2'/\Psi_2\} =_{\Psi_1 * \Psi_2} \chi'$.

- (SUB-LIFT). In this case we have $\sigma' \Downarrow \chi \colon \Gamma' \to \Delta'$ and $\sigma' \Downarrow \chi' \colon \Phi' \to \Psi'$ where $\sigma = \sigma'_\bullet$, $\Gamma = (\Gamma', x \colon A[\sigma'])$, $\Delta = (\Delta', x \colon A)$, $\Phi = (\Phi', y \colon B[\sigma'])$ and $\Psi = (\Psi', y \colon B)$. The (implicit) side-condition on (SUB-LIFT) gives $x \notin \Gamma', \Delta'$ and $y \notin \Phi', \Psi'$.

  From the assumption $v(\Gamma) = v(\Phi)$ it follows that $v(\Gamma') = v(\Phi')$ and $x = y$ hold. By induction hypothesis, we get an α satisfying $v(\Psi') = v(\Delta'\ \alpha^{-1})$ and $\alpha \circ \chi =_{\Psi'} \chi'$. Since $x \notin \Delta' \cup \Psi'$ and $\alpha(FV(\Psi')) = FV(\Delta')$ hold, there exists a bijective renaming β that agrees with α on $\Delta' \cup \Psi'$ and for which $\beta(x) = x$ holds. Therefore, we have $\beta \circ \chi =_{\Psi'} \chi'$. Using Lemma 6.4.6.4, we get $(\beta \circ \chi)(x) = x$ and $\chi'(x) = x$. Hence, we can conclude $\beta \circ \chi =_{\Psi', x \colon B} \chi'$, as required.

- (SUB-SUB). In this case we have $\Gamma \vdash M \Downarrow M' : A$, $\Phi \vdash M \Downarrow M'' : B$, Δ is $(\Gamma, x \colon A)$, Ψ is $(\Phi, y \colon B)$, χ is $\{M'/x\}$ and χ' is $\{M''/y\}$. By assumption $v(\Gamma) = v(\Phi)$ holds, so that we can use point $3'$ to obtain $M' = M''$. The assertion can then be satisfied by taking α to be the swapping $(x\ y)$.

- ($*$-E). As for (SUB-SUB).

$2'$. Suppose we have derivations $\Gamma \vdash A \Downarrow A'$ Type and $\Delta \vdash A \Downarrow A''$ Type with $v(\Gamma) = v(\Delta)$. The rules are syntax-directed, so that we can assume that both derivations end in the same last rule. The proof continues similarly to $3'$.

$3'$. Suppose we have derivations $\Gamma \vdash M \Downarrow M' : A$ and $\Delta \vdash M \Downarrow M'' : B$ with $v(\Gamma) = v(\Delta)$. If either derivation ends in (BU-CONV), then, by Lemma 6.4.7, we can immediately use the induction hypothesis. If either derivation has (TY-CONV) as its last rule then we get the assertion directly from the induction hypothesis, since we only need to consider the term but not the type. As the other rules are syntax-directed, we can assume that both derivations end in the same last rule. Case distinction on this last rule.

- (SUBST). From the first derivation we get $\sigma \Downarrow \chi : \Gamma \to \Gamma_1$ and $\Gamma_1 \vdash M_1 \Downarrow M_1' : A_1$ and $M' = M_1'\ \chi$. From the other derivation we get $\sigma \Downarrow \xi : \Delta \to \Delta_1$ and $\Delta_1 \vdash M_1 \Downarrow M_1'' : B_1$ and $M'' = M_1''\ \xi$. By assumption, $v(\Gamma) = v(\Delta)$ holds. We need to show $M' = M''$. First, by $1'$ we get a bijective renaming α such that $v(\Gamma_1) = v(\Delta_1\ \alpha)$ and $\alpha \circ \chi =_{\Delta_1} \xi$ hold. By Lemma 6.4.5 we get a renamed derivation $(\Gamma_1\ \alpha^{-1}) \vdash M_1 \Downarrow M_1'\ \alpha^{-1} : A_1$ of the same size as the derivation of $\Gamma_1 \vdash M_1 \Downarrow M_1' : A_1$. Since $v(\Gamma_1\ \alpha^{-1}) = v(\Delta_1)$, we can apply the induction hypothesis to obtain $M_1'\ \alpha^{-1} = M_1''$. Since $\alpha \circ \chi =_{\Delta_1} \xi$ and because $FV(M_1'') \subseteq FV(\Delta_1)$, by Lemma 6.4.6, this implies $M_1'\ \alpha^{-1}\ (\alpha \circ \chi) = M_1''\ \xi$, i.e. $M' = M_1'\ \chi = M_1''\ \xi = M''$, as required.
- (C-TM). From the first derivation we get $\sigma \Downarrow \chi : \Gamma \to \Delta$ and $M' = f(x_1, \ldots, x_n)\ \chi$, where $v(\Delta)$ is $x_1 \ldots x_n$. From the other derivation we get $\sigma \Downarrow \xi : \Gamma \to \Phi$ and $N' = f(y_1, \ldots, y_m)\ \xi$, where $v(\Phi)$ is $y_1 \ldots y_m$.

  By induction hypothesis $1'$, we have α such that $v(\Phi) = v(\Delta\ \alpha^{-1})$ and $\alpha \circ \chi =_\Phi \xi$. We note that $f(y_1, \ldots, y_m)\ \alpha = f(x_1, \ldots, x_n)$. Therefore, $M' = f(x_1, \ldots, x_n)\ \chi = f(y_1, \ldots, y_m)\ \alpha\ \chi = f(y_1, \ldots, y_m)\ \xi = N'$, as required.
- ($*$-I), (Π-I), (Π-E), (Π^*-I), (Π^*-E) follow directly from the induction hypothesis.

To get the assertion of the lemma it now suffices to show that $1'$ implies 1. To this end, suppose $\sigma \Downarrow \chi : \Gamma \to \Delta$ and $\sigma \Downarrow \chi' : \Gamma \to \Delta$. By $1'$ we obtain a bijective renaming α satisfying $v(\Delta) = v(\Delta\ \alpha^{-1})$ and $\alpha \circ \chi =_\Delta \chi'$. Because of $v(\Delta) = v(\Delta\ \alpha^{-1})$, α must be the identity on $v(\Delta)$. This implies $\chi =_\Delta \chi'$, and from this we get the required $\chi = \chi'$ by Lemma 6.4.6.4. □

We define when two explicit substitutions denote the same syntactic substitution.

Definition 6.4.9. Define $\sigma \equiv \tau : \Gamma \to \Delta$ to hold whenever there exist χ and ξ such that $\sigma \Downarrow \chi : \Gamma \to \Delta$ and $\tau \Downarrow \xi : \Gamma \to \Delta$ are derivable and $\chi =_\Delta \xi$, as defined in (6.1), holds.

Because of the equality rules, provably equal explicit substitutions may translate to different syntactic substitutions. The converse, that explicit substitutions that translate to the same syntactic substitution are provably equal will follow from the Main Lemma 6.5.26.

Lemma 6.4.10. *The syntactic translations of substitutions have the following properties.*

1. *If* $\sigma \equiv \tau : \Gamma \to \Delta$ *and* $\sigma \Downarrow \chi : \Gamma \to \Delta$ *and* $\tau \Downarrow \xi : \Gamma \to \Delta$ *then* $\chi =_\Delta \xi$.
2. $(-) \equiv (-) : \Gamma \to \Delta$ *is an equivalence relation.*
3. *If* $\sigma \equiv \tau : \Gamma \to \Delta$ *and* $\vdash \Gamma = \Gamma'$ Bunch *and* $\vdash \Delta = \Delta'$ Bunch *then* $\sigma \equiv \tau : \Gamma' \to \Delta'$.
4. *If* $\sigma \equiv \tau : \Gamma \to \Delta$ *and* $\sigma' \equiv \tau' : \Delta \to \Phi$ *then* $\sigma' \circ \sigma \equiv \tau' \circ \tau : \Gamma \to \Phi$.
5. *If* $\sigma : \Gamma \to \Delta$ *and* $\sigma' : \Delta \to \Phi$ *and* $\sigma'' : \Phi \to \Psi$ *then* $(\sigma'' \circ \sigma') \circ \sigma \equiv \sigma'' \circ (\sigma' \circ \sigma) : \Gamma \to \Psi$.
6. *If* $\sigma \equiv \tau : \Gamma \to \Delta$ *and* $\sigma' \equiv \tau' : \Gamma' \to \Delta'$ *and we have* $\emptyset = (\Gamma \cup \Delta) \cap (\Gamma' \cup \Delta')$ *then we also have* $(\sigma * \sigma') \equiv (\tau * \tau') : \Gamma * \Gamma' \to \Delta * \Delta'$.

7. *If* $\sigma \equiv \tau : \Gamma \to \Delta$ *and* $\Delta \vdash A$ Type *and* $x \notin \Gamma \cup \Delta$ *then* $\sigma_\bullet \equiv \tau_\bullet : \Gamma, x\colon A[\sigma] \to \Delta, x\colon A$.

8. *If* $\sigma\colon \Gamma \to \Delta$ *and* $\tau\colon \Phi \to \Psi$ *then we have* $(\sigma * \tau) \equiv (\sigma * id) \circ (id * \tau)\colon \Gamma * \Phi \to \Delta * \Psi$*, provided that* $\emptyset = (\Gamma \cup \Delta) \cap (\Phi \cup \Psi)$ *holds.*

9. *If* $\Gamma \vdash M : A$ *then* $\mathrm{w} \circ \langle M \rangle \equiv id\colon \Gamma \to \Gamma$.

Proof. The first point follows from Lemma 6.4.8. The other points are straightforward properties of syntactic substitution. □

6.4.4 Interpretation and Soundness of ES

The explicit system **ES** is interpreted in a $(*, 1, \Sigma, \Pi, \Pi^*)$-type-category. The interpretation of the term and type constants in this category is given by a $(*, 1, \Sigma, \Pi, \Pi^*)$-structure, the definition of which is adapted from [85].

Definition 6.4.11. A $(*, 1, \Sigma, \Pi, \Pi^*)$-*structure* for a theory $\mathscr{T}$ is given by the following data.

1. A $(*, 1, \Sigma, \Pi, \Pi^*)$-type-category. We denote the fibration by $p\colon \mathbb{E} \to \mathbb{B}$.

2. For each type constant T in $\mathscr{T}(\Gamma)$, an object Γ_T in $\mathbb{B}$ and an object A_T in $\mathbb{E}_{\Gamma_T}$.

3. For each term constant f in $\mathscr{T}(\Gamma; A)$, an object Γ_f in $\mathbb{B}$, an object A_f in $\mathbb{E}_{\Gamma_f}$ and a morphism $M_f\colon 1 \to A_f$ in $\mathbb{E}_{\Gamma_f}$.

To define the interpretation of the syntax in this structure, we first define an, a priori partial, interpretation. It then follows by a straightforward induction on the derivations that this defines a total function and a sound interpretation. Because the type system **ES** has explicit substitutions and structural rules, we do not need to establish weakening and substitution lemmas before showing correctness, as is necessary in [105, 85].

The interpretation is defined by induction on the derivations. Since it is mainly standard, we only give the cases for $*$-types and Π^*-types and some cases for substitutions here. In the following definitions, we follow the convention that if an expression has an undefined subexpression, or if the types of an expression do not match the requirements, then the whole expression is undefined. We write $!_B\colon B \to 1$ for the terminal map. Given an object Γ in $\mathbb{B}$ and an object A in $\mathbb{E}_1$, we write $weak_{\Gamma,A}\colon \{!_\Gamma^* A\} \to \{A\}$ for the morphism in $\mathbb{B}$ given by $\{\overline{!_\Gamma(A)}\}$.

($*$-Ty). $\|\Gamma; A * B\| = !^*_{\|\Gamma\|}(\|\diamond; A\| \overline{*} \|\diamond; B\|)$

($*$-I). $\|\Gamma_1 * \Gamma_2; \mathrm{pair}^*_{A,B}(M,N)\| = s$ where $s\colon 1_{\|\Gamma_1 * \Gamma_2\|} \to A' \overline{*} B'$ is the unique map satisfying

$$\{\overline{!_{\|\Gamma_1 * \Gamma_2\|}}(A' \overline{*} B')\} \circ \{s\} = l_{A',B'} \circ ((\{M'\} \circ weak_{\|\Gamma_1\|,A'}) * (\{N'\} \circ weak_{\|\Gamma_2\|,B'})),$$

provided that M' is $\|\Gamma_1; M\|$, N' is $\|\Gamma_2; N\|$, A' is $\|\diamond; A\|$ and B' is $\|\diamond; B\|$.

(Π^*-I). $\|\Gamma;\lambda^* A.M\| = \lambda^*\|\Gamma * x\colon A;M\|$ where x is some/any fresh variable.

(Π^*-E). $\|\Gamma_1 * \Gamma_2; \mathrm{app}^*_{[A]B}(M,N)\| = \left(id_{\|\Gamma_1\|} * (\{\|\Gamma_2;N\|\} \circ weak_{\|\Gamma_2\|,\|\diamond;A\|})\right)^* (\mathrm{app}^*\|\Gamma_1;M\|)$

(STR-ID). $\|id\colon \Gamma \to \Delta\| = id_{\|\Delta\|}$

(STR-WEAK). $\|\mathrm{w}\colon (\Gamma, x\colon A) \to \Delta\| = \pi_{\|\Gamma;A\|}$

(STR-$*$). $\|\sigma * \tau\colon \Gamma_1 * \Gamma_2 \to \Delta_1 * \Delta_2\| = \|\sigma\colon \Gamma_1 \to \Delta_1\| * \|\tau\colon \Gamma_2 \to \Delta_2\|$

(SUB-LIFT). $\|\sigma_\bullet\colon \Gamma, x\colon A \to \Delta, x\colon B\| = \{\overline{\|\sigma\colon \Gamma \to \Delta\|}(\|\Delta;B\|)\}$

(SUB-SUB). $\|\langle M\rangle\colon \Gamma \to \Delta, x\colon A\| = \{\|\Delta;M\|\}$

($*$-E). $\|\langle M/A*B\rangle\colon \Gamma \to (x\colon C) * (y\colon D)\| = l^{-1}_{\|\diamond;A\|,\|\diamond;A\|} \circ \{!_{\|\Gamma\|}(\|\diamond;A\| \overline{*} \|\diamond;B\|) \circ \|\Gamma;M\|\}$

Definition 6.4.12. A $(*,1,\Sigma,\Pi,\Pi^*)$-structure is a $(*,1,\Sigma,\Pi,\Pi^*)$-*model* for $\mathscr{T}$ if the following hold.

1. For each type constant T in $\mathscr{T}(\Gamma)$ for which $\vdash_{\mathbf{ES}} \Gamma$ Bunch is derivable, the interpretation $\|\Gamma\|$ is defined and equal to Γ_T.
2. For each term constant f in $\mathscr{T}(\Gamma;A)$ for which $\Gamma \vdash_{\mathbf{ES}} A$ Type is derivable, the interpretations $\|\Gamma\|$ and $\|\Gamma;A\|$ are defined and equal to Γ_f and A_f respectively.
3. For each axiom $\Gamma \vdash_{\mathbf{ES}} A = B$ Type in $\mathscr{T}^{\mathscr{E}}(\Gamma)$ for which both $\Gamma \vdash_{\mathbf{ES}} A$ Type and $\Gamma \vdash_{\mathbf{ES}} B$ Type are derivable, both $\|\Gamma;A\|$ and $\|\Gamma;B\|$ are defined and denote the same object of $\mathbb{E}_{\|\Gamma\|}$.
4. For each axiom $\Gamma \vdash_{\mathbf{ES}} M = N : A$ in $\mathscr{T}^{\mathscr{E}}(\Gamma;A)$ for which $\Gamma \vdash_{\mathbf{ES}} M : A$ and $\Gamma \vdash_{\mathbf{ES}} M : A$ are derivable, both $\|\Gamma;M\|$ and $\|\Gamma;N\|$ are defined and denote the same morphism of type $1 \to \|\Gamma;A\|$ in $\mathbb{E}_{\|\Gamma\|}$.

Proposition 6.4.13 (Soundness)**.** *For any $(*,1,\Sigma,\Pi,\Pi^*)$-model, the following hold.*

- *If* $\vdash_{\mathbf{ES}} \Gamma$ Bunch *then* $\|\Gamma\|$ *is an object of* $\mathbb{B}$.
- *If* $\Gamma \vdash_{\mathbf{ES}} A$ Type *then* $\|\Gamma\|$ *is an object of* $\mathbb{B}$ *and* $\|\Gamma;A\|$ *is an object of* $\mathbb{E}_{\|\Gamma\|}$.
- *If* $\Gamma \vdash_{\mathbf{ES}} M : A$ *then* $\|\Gamma\|$ *is an object of* $\mathbb{B}$, $\|\Gamma;A\|$ *is an object of* $\mathbb{E}_{\|\Gamma\|}$ *and* $\|\Gamma;M\|$ *is a map of type* $1_{\|\Gamma\|} \to \|\Gamma;A\|$ *in* $\mathbb{E}_{\|\Gamma\|}$.
- *If* $\sigma\colon \Gamma \to \Delta$ *or* $\sigma\colon \Gamma \Rightarrow \Delta$ *then* $\|\Gamma\|$ *and* $\|\Delta\|$ *are objects of* $\mathbb{B}$ *and* $\|\sigma\colon \Gamma \to \Delta\|$ *is a map of type* $\|\Gamma\| \to \|\Delta\|$ *in* $\mathbb{B}$.
- *If* $\vdash_{\mathbf{ES}} \Gamma = \Delta$ Bunch *then* $\|\Gamma\|$ *and* $\|\Delta\|$ *are the same object of* $\mathbb{B}$.
- *If* $\Gamma \vdash_{\mathbf{ES}} A = B$ Type *then* $\|\Gamma\|$ *is an object of* $\mathbb{B}$ *and* $\|\Gamma;A\|$ *and* $\|\Gamma;B\|$ *are the same object of the fibre category* $\mathbb{E}_{\|\Gamma\|}$.
- *If* $\Gamma \vdash_{\mathbf{ES}} M = N : A$ *then* $\|\Gamma\|$ *is an object of* $\mathbb{B}$, $\|\Gamma;A\|$ *is an object of* $\mathbb{E}_{\|\Gamma\|}$ *and* $\|\Gamma;M\|$ *and* $\|\Gamma;N\|$ *are the same map of type* $1_{\|\Gamma\|} \to \|\Gamma;A\|$ *in* $\mathbb{E}_{\|\Gamma\|}$.

Proof. By simultaneous induction on the derivations. □

6.5 Interpretation and Soundness of $\mathbf{BT}(*, \Pi, \Pi^*)$

In this section we define the translation from **BT** to **ES**. As explained at the beginning of this chapter, the main effort goes in the proof of a Main Lemma stating that two judgements in **ES** with the same translation in the let-free syntax are provably equal. The following lemmas contribute to the proof of this Main Lemma.

We start by considering the different ways of applying the structural rules after the projection rule, i.e. we consider derivations of judgements of the form $\Gamma \vdash 1[\delta] : A[\mathrm{w}][\delta]$ where δ is a structural map. Such sequents correspond to sequents of the form $\Gamma(\Delta, x\colon A) \vdash x : A$ in $\mathbf{BT}(*, \Pi, \Pi^*)$.

Lemma 6.5.1. *If $\sigma\colon \Gamma \Rightarrow \Delta, x\colon A$ and $\tau\colon \Gamma \Rightarrow \Phi, x\colon B$ then both $\Gamma \vdash A[\mathrm{w}][\sigma] = B[\mathrm{w}][\tau]$ Type and $\Gamma \vdash 1[\sigma] = 1[\tau] : A[\mathrm{w}][\sigma]$ are derivable.*

For the proof of this lemma we need some auxiliary notions. First, for any $\vdash \Gamma$ Bunch with $x \in \Gamma$, we define a structural morphism $\hat{x}_\Gamma : \Gamma \Rightarrow \Gamma(x), x\colon A(\Gamma, x)$ by induction on the derivation of $\vdash \Gamma$ Bunch.

- Γ cannot be $\diamond$ since $x \in \Gamma$.
- Γ is $\Gamma', y\colon A$. If $x = y$ then $\hat{x}_\Gamma = id : \Gamma \Rightarrow \Gamma$, and if $x \neq y$ then $\hat{x}_\Gamma = \hat{x}_{\Gamma'} \circ \mathrm{w}\colon \Gamma \to \Gamma'(x), x\colon A(\Gamma', x)$.
- Γ is $\Gamma' * \Gamma''$. If $x \in \Gamma'$ then $\hat{x}_\Gamma = \hat{x}_{\Gamma'} \circ \dot{\pi}_1$, and if $x \in \Gamma''$ then $\hat{x}_\Gamma = \hat{x}_{\Gamma''} \circ \dot{\pi}_2$.

It is immediate from this definition that if $\vdash \Gamma$ Bunch and $x \in \Gamma$ hold then $\Gamma(x) \vdash A(\Gamma, x)$ Type.

Lemma 6.5.2. *Let $\sigma : \Gamma \Rightarrow \Delta$ and $x \in \Delta$. Then there exists a structural map $\tau : \Gamma(x) \Rightarrow \Delta(x)$ such that the following diagram commutes.*

$$\begin{array}{ccc} \Gamma & \xRightarrow{\quad\sigma\quad} & \Delta \\ {\scriptstyle \hat{x}_\Gamma}\Downarrow & & \Downarrow{\scriptstyle \hat{x}_\Delta} \\ \Gamma(x), x\colon A(\Gamma, x) & \xRightarrow[\tau_\bullet]{} & \Delta(x), x\colon A(\Delta, x) \end{array}$$

Proof. The proof goes by induction on the derivation of $\sigma : \Gamma \Rightarrow \Delta$. We continue by case distinction on the last rule.

- ($\Rightarrow$-BU-CONV). We know that $\vdash \Gamma = \Gamma'$ Bunch, $\sigma\colon \Gamma' \Rightarrow \Delta'$ and $\vdash \Delta' = \Delta$ Bunch are derivable. From the bunch equality judgements, it is straightforward to show $\vdash \Gamma(x), x\colon A(\Gamma, x) = \Gamma'(x), x\colon A(\Gamma', x)$ Bunch and $\vdash \Delta(x), x\colon A(\Delta, x) = \Delta'(x), x\colon A(\Delta', x)$ Bunch. Using the induction hypothesis, we have

$$\begin{array}{ccccccc} \Gamma & = & \Gamma' & \xRightarrow{\quad\sigma\quad} & \Delta' & = & \Delta \\ {\scriptstyle \hat{x}_\Gamma}\Downarrow & & {\scriptstyle \hat{x}_{\Gamma'}}\Downarrow & & \Downarrow{\scriptstyle \hat{x}_{\Delta'}} & & \Downarrow{\scriptstyle \hat{x}_\Delta} \\ \Gamma(x), x\colon A(\Gamma, x) & = & \Gamma'(x), x\colon A(\Gamma', x) & \xRightarrow[\tau_\bullet]{} & \Delta'(x), x\colon A(\Delta', x) & = & \Delta(x), x\colon A(\Delta, x) \end{array} .$$

  The outermost vertical maps are typeable using ($\Rightarrow$-BU-CONV). Therefore, we can use the rule ($\Rightarrow$-BU-CONV) with $\tau_\bullet$ to get the required result.

- (STR-ID). We know that σ is *id*, that $\vdash \Gamma$ Bunch is derivable and that Γ and Δ are identical. Therefore, we take τ to be $id : \Gamma(x) \Rightarrow \Gamma(x)$.

- (STR-COMP). We have that σ is $\sigma'' \circ \sigma'$, where $\sigma' : \Gamma \Rightarrow \Phi$ and $\sigma'' : \Phi \Rightarrow \Delta$. The induction hypothesis yields

$$\begin{array}{ccc} \Gamma & \xrightarrow{\sigma'} & \Phi \\ {\scriptstyle \hat{x}_\Gamma}\Downarrow & & \Downarrow{\scriptstyle \hat{x}_\Phi} \\ \Gamma(x), x\colon A(\Gamma,x) & \xrightarrow[\tau'_\bullet]{} & \Phi(x), x\colon A(\Phi,x) \end{array} \qquad \begin{array}{ccc} \Phi & \xrightarrow{\sigma''} & \Delta \\ {\scriptstyle \hat{x}_\Phi}\Downarrow & & \Downarrow{\scriptstyle \hat{x}_\Delta} \\ \Phi(x), x\colon A(\Phi,x) & \xrightarrow[\tau''_\bullet]{} & \Delta(x), x\colon A(\Delta,x) \end{array} .$$

  We can paste these diagrams using $(\tau'' \circ \tau')_\bullet = \tau''_\bullet \circ \tau'_\bullet$.

- (STR-$*$). In this case, σ is $\sigma' * \sigma''$ where $\sigma' : \Gamma' \Rightarrow \Delta'$ and $\sigma'' : \Gamma'' \Rightarrow \Delta''$. Without loss of generality assume that $x \in FV(\Delta') \subseteq FV(\Gamma')$ holds. Using the induction hypothesis, we have:

$$\begin{array}{ccc} \Gamma & \xrightarrow{\sigma' * \sigma''} & \Delta \\ {\scriptstyle \dot{\pi}_1}\Downarrow & & \Downarrow{\scriptstyle \dot{\pi}_1} \\ \Gamma' & \xrightarrow{\sigma'} & \Delta' \\ {\scriptstyle \hat{x}_{\Gamma'}}\Downarrow & & \Downarrow{\scriptstyle \hat{x}_{\Delta'}} \\ \Gamma'(x), x\colon A(\Gamma',x) & \xrightarrow[\tau'_\bullet]{} & \Delta'(x), x\colon A(\Delta',x) \end{array}$$

  The upper square is easily constructed, and the lower square follows from the induction hypothesis. Since the vertical composites are just $\hat{x}_\Gamma$ and $\hat{x}_\Delta$, we are done.

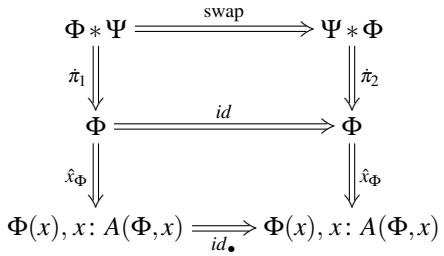
- (STR-LIFT). We know that σ is $\sigma'_\bullet$. There are two possible cases. First, Δ is $\Delta', x\colon A$. In this case, both $\hat{x}_\Gamma$ and $\hat{x}_\Delta$ are the identity, so by taking $\tau = id$ we are done. If Δ is $\Delta', y\colon A$ for $y \neq x$, then we continue using the following diagram, the lower half of which is given by the induction hypothesis.

$$\begin{array}{ccc} \Gamma, y\colon B & \xrightarrow{\sigma'_\bullet} & \Delta, y\colon A \\ {\scriptstyle \mathrm{w}}\Downarrow & & \Downarrow{\scriptstyle \mathrm{w}} \\ \Gamma & \xrightarrow{\sigma'} & \Delta \\ {\scriptstyle \hat{x}_\Gamma}\Downarrow & & \Downarrow{\scriptstyle \hat{x}_\Delta} \\ \Gamma(x), x\colon A(\Gamma,x) & \xrightarrow[\tau'_\bullet]{} & \Delta(x), x\colon A(\Delta,x) \end{array}$$

  Again, note that the vertical maps are just the projections $\hat{x}_{(\Gamma, y\colon B)}$ and $\hat{x}_{(\Delta, y\colon A)}$. The upper diagram commutes by definition of the equations for substitution.

- (STR-WEAK). In this case σ is w, and we can take τ to be the identity.

- (STR-SWAP). The structural map σ is swap and the contexts Γ and Δ are of the form $\Phi * \Psi$ and $\Psi * \Phi$ respectively. Without loss of generality we may assume $x \in FV(\Phi) \subseteq FV(\Gamma)$. Then we have:

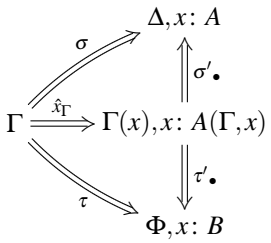


  That the upper square commutes follows easily from the coherence assumptions in the definition of symmetric monoidal categories, see e.g. the proof of coherence theorem for symmetric monoidal categories [63]. The lower square commutes because of the rule $((-)_\bullet$-ID). The assertion follows because the vertical composites in the above diagram are just $\hat{x}_{\Phi * \Psi}$ and $\hat{x}_{\Psi * \Phi}$ respectively.

- The remaining cases for (STR-UNIT), (STR-UNIT^{-1}), (STR-ASSOC) and (STR-ASSOC^{-1}) are similar to the case for (STR-SWAP).

□

Proof of Lemma 6.5.1. This proof now follows directly from the previous lemma. First note that $\hat{x}_{(\Delta, x\colon A)}$ and $\hat{x}_{(\Phi, x\colon B)}$ are both the identity. Hence, by the lemma, there exist maps σ' and τ' making the following diagram commute.



Then we have $\Gamma(x), x\colon A(\Gamma,x) \vdash A[\mathrm{w}][\sigma'_\bullet] = A[\sigma'][\mathrm{w}] = A(\Gamma,x)[\mathrm{w}] = B[\tau'][\mathrm{w}] = B[\mathrm{w}][\tau'_\bullet]$ Type. In this step we use the fact that if $\rho_\bullet\colon \Phi, x\colon B \Rightarrow \Psi, x\colon A$ is derivable then so is $\Phi \vdash B = A[\rho]$ Type, as established in Lemma 6.4.4. In the above case, we can thus derive $\Gamma(x) \vdash A[\sigma'] = A(\Gamma,x)$ Type. We furthermore have $\Gamma(x), x\colon A(\Gamma,x) \vdash 1[\sigma'_\bullet] = 1 : A(\Gamma,x)[\mathrm{w}]$ and $\Gamma(x), x\colon A(\Gamma,x) \vdash 1[\tau'_\bullet] = 1 : A(\Gamma,x)[\mathrm{w}]$. Substituting these equations along $\hat{x}_\Gamma$ finishes the proof:

$$\Gamma \vdash A[\mathrm{w}][\sigma] = A[\mathrm{w}][\sigma'_\bullet][\hat{x}_\Gamma] = B[\mathrm{w}][\tau'_\bullet][\hat{x}_\Gamma] = B[\mathrm{w}][\tau] \text{ Type}$$

$$\Gamma \vdash 1[\sigma] = 1[\sigma'_\bullet][\hat{x}_\Gamma] = 1[\hat{x}_\Gamma] = 1[\tau'_\bullet][\hat{x}_\Gamma] = 1[\tau] \ : \ A(\Gamma,x)[\mathrm{w}][\hat{x}_\Gamma] = A[\mathrm{w}][\sigma]$$

□

We obtain the coherence lemma that any two structural morphism of the same type are equal.

Lemma 6.5.3. *The following rule is admissible.*

$$(\textsc{Str-Eq})\ \frac{\sigma\colon \Gamma \Rightarrow \Delta \qquad \tau\colon \Gamma \Rightarrow \Delta}{\sigma = \tau\colon \Gamma \to \Delta}$$

Proof. By induction on the structure of the bunch Δ.

- Δ is $\diamond$. Immediate by (Sub-$\diamond$Eq).
- Δ is $\Delta', x\colon C$. By (Sub-AEq) it suffices to show $\mathrm{w}\circ\sigma = \mathrm{w}\circ\tau\colon \Gamma \Rightarrow \Delta'$ and $1[\sigma] = 1[\tau]\colon C[\mathrm{w}][\sigma] = C[\mathrm{w}][\tau]$. The first point follows from the induction hypothesis. The second point follows from Lemma 6.5.1.
- Δ is $\Delta_1 * \Delta_2$. By (Sub-MEq) it suffices to show $\dot{\pi}_i \circ \sigma = \dot{\pi}_i \circ \tau\colon \Gamma \to \Delta_i$, where $\dot{\pi}_i\colon \Delta_1 * \Delta_2 \Rightarrow \Delta_i$, for all $i \in \{1,2\}$. Both assertions follow from the induction hypothesis.

□

We should point out that this lemma is much easier to prove than the coherence theorem for symmetric monoidal categories [63] because of the assumption that $*$ is strict affine. Note also that this lemma makes use of the named variables in the contexts. Using the names in the contexts we obtain that two structural morphisms of the same type automatically have the same graph, in the sense of categorical coherence [59]. For example, the named variables are essential in order to avoid identifying $\mathrm{swap}\colon (x\colon A) * (y\colon A) \Rightarrow (y\colon A) * (x\colon A)$ and $id\colon (x\colon A) * (y\colon A) \Rightarrow (x\colon A) * (y\colon A)$ in this lemma.

The next lemma establishes a form of extensionality for substitutions.

Lemma 6.5.4. *Let $\sigma\colon \Gamma \to \Delta$ and $\tau\colon \Gamma \to \Delta$. If, for all structural maps $\delta\colon \Delta \Rightarrow \Phi, x\colon A$, we can derive $\Gamma \vdash 1[\delta][\sigma] = 1[\delta][\tau] : A[\mathrm{w}][\delta][\sigma] = A[\mathrm{w}][\delta][\tau]$, then $\sigma = \tau\colon \Gamma \to \Delta$ is derivable.*

Proof. By induction on the structure of the bunch Δ.

- Δ is $\diamond$. Immediate by (Sub-$\diamond$Eq).
- Δ is $\Delta', x\colon C$. By rule (Sub-AEq), it suffices to show $\mathrm{w}\circ\sigma = \mathrm{w}\circ\tau\colon \Gamma \to \Delta'$ and $1[\sigma] = 1[\tau]\colon C[\mathrm{w}][\sigma] = C[\mathrm{w}][\tau]$. For the first point we use the induction hypothesis as follows. Assume an arbitrary structural map $\delta\colon \Delta' \Rightarrow \Phi, x\colon A$. For each such δ we have $\delta\circ\mathrm{w}\colon \Delta \Rightarrow \Phi, x\colon A$. By assumption we have $\Gamma \vdash 1[\delta\circ\mathrm{w}][\sigma] = 1[\delta\circ\mathrm{w}][\tau] : A[\mathrm{w}][\delta\circ\mathrm{w}][\sigma] = A[\mathrm{w}][\delta\circ\mathrm{w}][\tau]$, which gives $\Gamma \vdash 1[\delta][\mathrm{w}\circ\sigma] = 1[\delta][\mathrm{w}\circ\tau] : A[\mathrm{w}][\delta][\mathrm{w}\circ\sigma] = A[\mathrm{w}][\delta][\mathrm{w}\circ\tau]$. Since δ was arbitrary, we can use the induction hypothesis to conclude $\mathrm{w}\circ\sigma = \mathrm{w}\circ\tau\colon \Gamma \to \Delta'$, as required. The second point follows from the assumption by taking $\delta = id$.
- Δ is $\Delta_1 * \Delta_2$. By (Sub-MEq) it suffices to show $\dot{\pi}_i \circ \sigma = \dot{\pi}_i \circ \tau\colon \Gamma \to \Delta_i$, where $\dot{\pi}_i\colon \Delta_1 * \Delta_2 \Rightarrow \Delta_i$, for all $i \in \{1,2\}$. Both assertions follow from the induction hypothesis as in the previous case.

□

Next we define a measure on types, terms and substitutions. The proof of the Main Lemma goes by induction on this measure.

Size of types:

$$\begin{aligned} m(A[\sigma]) &= m(A) + m(\sigma) \\ m(T(\sigma)) &= 2 + m(\sigma) \\ m(A * B) &= 1 + m(A) + m(B) \\ m(\Pi A.B) &= 1 + m(A) + m(B) \\ m(\Pi^* A.B) &= 1 + m(A) + m(B) \end{aligned}$$

Size of terms:

$$\begin{aligned} m(M[\sigma]) &= m(M) + m(\sigma) \\ m(1) &= 1 \\ m(f(\sigma)) &= 2 + m(\sigma) \\ m(\mathrm{pair}^*_{A,B}(M,N)) &= 1 + m(M) + m(A) + m(N) + m(B) \\ m(\lambda A.M) &= 1 + m(A) + m(M) \\ m(\mathrm{app}_{[A]B}(M,N)) &= 1 + m(A) + m(B) + m(M) + m(N) \\ m(\lambda^* A.M) &= 1 + m(A) + m(M) \\ m(\Pi^* A.B) &= 1 + m(A) + m(B) \\ m(\mathrm{app}^*_{[A]B}(M,N)) &= 1 + m(A) + m(B) + m(M) + m(N) \end{aligned}$$

Size of substitutions:

$$\begin{aligned} m(\tau : \Gamma \Rightarrow \Delta) &= 0 \\ m(\sigma \circ \tau) &= m(\sigma) + m(\tau) \\ m(\langle M \rangle) &= 1 + m(M) \\ m(\sigma_\bullet) &= m(\sigma) \\ m(\sigma * \tau) &= m(\sigma) + m(\tau) \\ m(\langle M/A * B \rangle) &= 1 + m(M) + m(A) + m(B) \end{aligned}$$

Note that the measure 'ignores' all structural maps.

In order to relate explicit substitution to normal syntactic substitution, we must show that the equations for substitutions $\sigma\colon \Gamma \to \Delta$ in the explicit calculus are sufficient to implement syntactic substitution. For example, the equation $1[\mathrm{w}][\langle M \rangle] = 1$ corresponds to the substitution property $x[M/y] = x$ where $x \neq y$, while $1[\langle M \rangle] = M$ corresponds to $x[M/x] = M$. In addition to the familiar equations for substitution, we must manage the structural morphisms. As in the examples just mentioned, depending

on the form of a structural map δ, the substitution $\langle M\rangle$ in the term $1[\delta\circ\langle M\rangle]$ may or may not be vacuous. We must be able to transform the substitution $\delta\circ\langle M\rangle$ into a form that makes it possible either to perform the substitution or to remove it if it is vacuous. We characterise such a special form of substitution by the following judgement $\sigma\colon\Gamma\xrightarrow{*}\Delta$. In this judgement, no structural map δ is allowed to be 'in the way' of the substitution $\langle M\rangle$, as is the case in $\delta\circ\langle M\rangle$.

$$(\textsc{Sub}^{+}\text{-}\textsc{Sub})\ \frac{\tau\colon\Gamma\Rightarrow\Delta\qquad\Delta\vdash A\ \mathrm{Type}\qquad\Gamma\vdash M:A[\tau]}{\langle\tau,M\rangle\colon\Gamma\xrightarrow{+}(\Delta,x\colon A)}$$

$$(\textsc{Sub}^{+}\text{-}\textsc{Let})\ \frac{\Gamma\vdash M:A*B\qquad\tau\colon(x\colon A)*(y\colon B)\Rightarrow\Delta}{\tau\circ\langle M/A*B\rangle\colon\Gamma\xrightarrow{+}\Delta}$$

$$(\textsc{Sub}^{+}\text{-}\textsc{Lift})\ \frac{\sigma\colon\Gamma\xrightarrow{+}\Delta\qquad\Delta\vdash A\ \mathrm{Type}\qquad\Gamma\vdash A[\sigma]=B\ \mathrm{Type}}{\sigma_{\bullet}\colon(\Gamma,x\colon B)\xrightarrow{+}(\Delta,x\colon A)}$$

$$(\textsc{Sub}^{+}\text{-}{*}\mathrm{L})\ \frac{\sigma\colon\Gamma\xrightarrow{+}\Delta\qquad\vdash\Phi\ \mathrm{Bunch}}{(id*\sigma)\colon\Phi*\Gamma\xrightarrow{+}\Phi*\Delta}\ \Phi\cap(\Gamma\cup\Delta)=\emptyset$$

$$(\textsc{Sub}^{+}\text{-}{*}\mathrm{R})\ \frac{\sigma\colon\Gamma\xrightarrow{+}\Delta\qquad\vdash\Phi\ \mathrm{Bunch}}{(\sigma*id)\colon\Gamma*\Phi\xrightarrow{+}\Delta*\Phi}\ \Phi\cap(\Gamma\cup\Delta)=\emptyset$$

$$(\textsc{Sub}^{*}\text{-}\textsc{Str})\ \frac{\tau\colon\Gamma\Rightarrow\Delta}{\tau\colon\Gamma\xrightarrow{*}\Delta}\qquad(\textsc{Sub}^{*}\text{-}\textsc{Comp})\ \frac{\sigma\colon\Gamma\xrightarrow{*}\Delta\qquad\tau\colon\Delta\xrightarrow{+}\Phi}{\tau\circ\sigma\colon\Gamma\xrightarrow{*}\Phi}$$

The judgement $\xrightarrow{*}$ is used to express substitutions in a form in which all of the structural maps have been 'moved out of the way'. In general, a morphism $\xrightarrow{*}$ has the form $\Rightarrow\xrightarrow{+}\ldots\xrightarrow{+}$ for zero or more $\xrightarrow{+}$. Each morphism $\xrightarrow{+}$ contains either a substitution or a let as its main part.

Definition 6.5.5 (Main part)**.** Let $\sigma\colon\Gamma\xrightarrow{+}\Delta$. It is a lifting via an iteration of $(-)*id$, $id*(-)$ and $(-)_{\bullet}$ of either $\sigma'\circ\langle M/A*B\rangle$ or $\langle\sigma',M\rangle$, which we call the *main part* of σ.

Our next goal is to show that each $\rightarrow$-substitution σ can be transformed into a $\xrightarrow{*}$-substitution τ, which amounts to moving structural maps out of the way. Moreover, we show that the translation is such that the measure does not increase, i.e. $m(\sigma)\geq m(\tau)$ holds, and both substitutions denote the same syntactic substitution $\sigma\equiv\tau$.

Some words are in order on the choice of $\xrightarrow{*}$ as a 'semi-canonical form' of substitutions. In standard dependent type theory, substitutions can be brought into a simple normal form in which substitutions can be built without the need for a composition rule, see [85, 48, 110]. In the presence of morphisms of the form $\langle M/A*B\rangle\colon\Gamma\rightarrow(x\colon A)*(y\colon B)$ as well as the structural maps for $*$, however, such a simple normal form does not appear to be possible. Indeed, in the proof of coherence for symmetric monoidal categories [63] composites of the coherent isomorphisms have relatively complicated canonical forms. Since these coherent isomorphisms are part of our structural maps, true canonical forms are likely to be complicated.

Lemma 6.5.6. *For each* $\sigma\colon \Gamma \xrightarrow{*} \Delta$ *we can derive* $\sigma\colon \Gamma \to \Delta$*, where we consider* $\langle \tau, M\rangle$ *as an abbreviation for* $\tau_\bullet \circ \langle M\rangle$*.*

Proof. Straightforward induction. □

Justified by this lemma, we use the measure m for $\xrightarrow{+}$ and $\xrightarrow{*}$-substitutions. For example, $m(\langle \tau, M\rangle) = 1 + m(\tau) + m(M)$. Also, we will sometimes write $\sigma \Downarrow \chi\colon \Gamma \xrightarrow{+} \Delta$ or $\sigma \Downarrow \chi\colon \Gamma \xrightarrow{*} \Delta$, where χ denotes the unique syntactic substitution obtained by the translation in the lemma.

Lemma 6.5.7. *For all* $\sigma\colon \Gamma \xrightarrow{+} \Delta$ *we have* $m(\sigma) \geq 1$*.*

Lemma 6.5.8 (Closure under bijective renaming)**.** *Let* α *be a bijective renaming.*

1. *If* $\sigma\colon \Gamma \xrightarrow{+} \Delta$ *is derivable then so is* $\sigma\colon \Gamma\,\alpha \xrightarrow{+} \Delta\,\alpha$*.*
2. *If* $\sigma\colon \Gamma \xrightarrow{*} \Delta$ *is derivable then so is* $\sigma\colon \Gamma\,\alpha \xrightarrow{*} \Delta\,\alpha$*.*

Moreover, the obtained derivations differ from the original ones only in the names of variables, but not in form or size.

Proof. Immediate, all rules are closed under bijective renaming. □

Lemma 6.5.9.

1. *If* $\sigma\colon \Gamma \xrightarrow{+} \Delta$ *and* $\vdash \Gamma' = \Gamma$ Bunch *and* $\vdash \Delta = \Delta'$ Bunch *then* $\sigma\colon \Gamma' \xrightarrow{+} \Delta'$*.*
2. *If* $\sigma\colon \Gamma \xrightarrow{*} \Delta$ *and* $\vdash \Gamma' = \Gamma$ Bunch *and* $\vdash \Delta = \Delta'$ Bunch *then* $\sigma\colon \Gamma' \xrightarrow{*} \Delta'$*.*

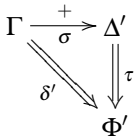
Proof. The first point follows by induction on the derivations of $\sigma\colon \Gamma \xrightarrow{+} \Delta$ and $\vdash \Gamma' = \Gamma$ Bunch, and point 2 is a consequence of 1. □

Lemma 6.5.10. *For all* $\sigma\colon \Gamma \xrightarrow{+} \Delta$ *and all* $\tau\colon \Delta \Rightarrow \Phi$ *one of the following is true:*

1. *There exists* $\delta\colon \Gamma \Rightarrow \Phi$ *such that* $\tau \circ \sigma = \delta$ *and* $\tau \circ \sigma \equiv \delta\colon \Gamma \to \Phi$ *hold; or*
2. *There exist* $\delta\colon \Gamma \Rightarrow \Psi$ *and* $\rho\colon \Psi \xrightarrow{+} \Phi$*, such that* $\tau \circ \sigma = \rho \circ \delta\colon \Gamma \to \Phi$*,* $m(\sigma) \geq m(\rho)$ *and* $\tau \circ \sigma \equiv \rho \circ \delta\colon \Gamma \to \Phi$ *all hold.*

Proof. The proof goes by induction on the derivation of τ.

- The last rule is ($\Rightarrow$-BU-CONV). We know that $\vdash \Delta = \Delta'$ Bunch, $\tau\colon \Delta' \Rightarrow \Phi'$ and $\vdash \Phi' = \Phi$ Bunch are derivable. Let $\sigma\colon \Gamma \xrightarrow{+} \Delta$. Then, by Lemma 6.5.9, $\sigma\colon \Gamma \xrightarrow{+} \Delta'$, so that we can apply the induction hypothesis to obtain one of two possible situations. First we obtain δ' such that the diagram



commutes and $\tau \circ \sigma \equiv \delta' \colon \Gamma \to \Phi'$ holds. Using ($\Rightarrow$-BU-CONV), we also have $\delta \colon \Gamma \Rightarrow \Phi$, and using ($\to$-EQ-BU-CONV) also $\tau \circ \sigma = \delta \colon \Gamma \to \Phi$. By Lemma 6.4.10.3, we also get $\tau \circ \sigma \equiv \delta \colon \Gamma \to \Phi$, so that 1 is satisfied. Second, we obtain ρ and δ such that the diagram

$$\begin{array}{ccc} \Gamma & \xrightarrow[\sigma]{+} & \Delta' \\ {\scriptstyle \delta'}\Downarrow & & \Downarrow{\scriptstyle \tau} \\ \Theta & \xrightarrow[\rho]{+} & \Phi' \end{array}$$

commutes, and $m(\sigma) \geq m(\rho)$ and $\tau \circ \sigma \equiv \rho \circ \delta \colon \Gamma \to \Phi'$ hold. Using Lemma 6.5.9 we also get $\rho \colon \Theta \xrightarrow{+} \Phi$, and 2 follows by conversion as before.

- τ is id. Satisfy 2 with $\rho = \sigma$ and $\delta = id$.
- τ is $\tau'' \circ \tau' : \Delta \Rightarrow \Psi \Rightarrow \Phi$ We apply the induction hypothesis to τ' and σ. There are two cases to consider. First, we obtain δ' such that the diagram

$$\begin{array}{ccc} \Gamma & \xrightarrow[\sigma]{+} & \Delta \\ & {\scriptstyle \delta'}\searrow & \Downarrow{\scriptstyle \tau'} \\ & & \Psi \end{array}$$

commutes and $\tau' \circ \sigma \equiv \delta' \colon \Gamma \to \Psi$ holds. In this case we let $\delta = \tau'' \circ \delta'$. We get $\tau \circ \sigma \equiv \delta \colon \Gamma \to \Psi$ by Lemma 6.4.10.4. Thus 1 holds.

Second, we obtain ρ' and δ' such that the diagram

$$\begin{array}{ccc} \Gamma & \xrightarrow[\sigma]{+} & \Delta \\ {\scriptstyle \delta'}\Downarrow & & \Downarrow{\scriptstyle \tau'} \\ \Theta & \xrightarrow[\rho']{+} & \Psi \end{array}$$

commutes, and $m(\sigma) \geq m(\rho')$ and $\tau' \circ \sigma \equiv \rho' \circ \delta' \colon \Gamma \to \Psi$ hold. We can apply the induction hypothesis to ρ' and τ''. There are again two cases to consider. First, we obtain δ'' such that the diagram

$$\begin{array}{ccc} \Theta & \xrightarrow[\rho']{+} & \Psi \\ & {\scriptstyle \delta''}\searrow & \Downarrow{\scriptstyle \tau''} \\ & & \Psi \end{array}$$

commutes and $\tau'' \circ \rho' \equiv \delta'' \colon \Theta \to \Psi$ holds. Then we can satisfy 1 by letting $\delta = \delta'' \circ \delta'$. Second, we obtain ρ'' and δ'' such that the diagram

$$\begin{array}{ccc} \Theta & \xrightarrow[\rho']{+} & \Psi \\ {\scriptstyle \delta''}\Downarrow & & \Downarrow{\scriptstyle \tau''} \\ \Xi & \xrightarrow[\rho'']{+} & \Phi \end{array}$$

commutes, and $m(\rho') \geq m(\rho'')$ and $\tau'' \circ \rho' \equiv \rho'' \circ \delta'' \colon \Theta \to \Phi$ hold. We can satisfy 2 by letting $\rho = \rho''$ and $\delta = \delta'' \circ \delta'$. We have $\tau \circ \sigma = \tau'' \circ \tau' \circ \sigma = \tau'' \circ \rho' \circ \delta' = \rho'' \circ \delta'' \circ \delta' = \rho \circ \delta$, and the same for $\equiv$. Finally, we have $m(\sigma) \geq m(\rho') \geq m(\rho)$.

- τ is $\tau' * \tau'' \colon \Delta' * \Delta'' \Rightarrow \Phi' * \Phi''$. Case distinction on the last rule in the derivation of σ:
  - σ is $\langle \sigma', M \rangle \colon \Gamma \xrightarrow{+} \Delta, x \colon A$. Impossible because of the form of the codomain.
  - σ is $\sigma'_\bullet$. Impossible because of the form of the codomain.
  - σ is $\sigma' * id \colon \Gamma' * \Delta'' \xrightarrow{+} \Delta' * \Delta''$. We apply the induction hypothesis to τ' and σ'. There are two cases to consider. First, we obtain δ' making the following diagram commute.

$$\begin{array}{ccc} \Gamma' & \xrightarrow{+}_{\sigma'} & \Delta' \\ & {\scriptstyle \delta'} \searrow & \Big\Downarrow {\scriptstyle \tau'} \\ & & \Phi' \end{array}$$

    We can satisfy 1 by letting $\delta = \delta' * id$. Second, we obtain ρ' and δ' for which

$$\begin{array}{ccc} \Gamma' & \xrightarrow[\sigma']{+} & \Delta' \\ {\scriptstyle \delta'} \Big\Downarrow & & \Big\Downarrow {\scriptstyle \tau'} \\ \Theta' & \xrightarrow[\rho']{+} & \Phi' \end{array}$$

    commutes, and $m(\sigma') \geq m(\rho')$ and $\tau' \circ \sigma' \equiv \rho' \circ \delta' \colon \Gamma' \to \Phi'$ hold. Hence

$$\begin{array}{ccc} \Gamma' * \Delta'' & \xrightarrow[\sigma' * id]{+} & \Delta' * \Delta'' \\ {\scriptstyle \delta' * \tau''} \Big\Downarrow & & \Big\Downarrow {\scriptstyle \tau' * \tau''} \\ \Theta' * \Phi'' & \xrightarrow[\rho' * id]{+} & \Phi' * \Phi'' \end{array} .$$

    The components of this diagram are typeable, since from the derivations for σ and τ we get $(\Gamma' \cup \Delta') \cap \Delta'' = (\Delta' \cup \Phi') \cap (\Delta'' \cup \Phi'') = \emptyset$ and because by Lemma 6.4.6 we have $\Theta' \subseteq \Gamma'$, $\Phi' \subseteq \Delta'$ and $\Phi'' \subseteq \Delta''$.

    We take $\delta = \delta' * \tau''$ and $\rho = \rho' * id$. We have $\tau \circ \sigma = (\tau' * \tau'') \circ (\sigma' * id) = ((\tau' \circ \sigma') * id) \circ (id * \tau'') = ((\rho' \circ \delta') * id) \circ (id * \tau'') = (\rho' * id) \circ (\delta' * \tau'') = \rho \circ \delta$. Furthermore, since structural maps translate to the identity substitution, we get $(\tau' * \tau'') \circ (\sigma' * id) \equiv (\rho' \circ id) \circ (\delta' * \tau'') \colon \Gamma' * \Delta' \to \Phi' * \Phi''$. Finally, $m(\sigma) = m(\sigma') \geq m(\rho') = m(\rho)$ holds.
  - σ is $id * \sigma'' \colon \Delta' * \Gamma'' \xrightarrow{+} \Delta' * \Delta''$. Symmetric case.
  - σ is $\sigma' \circ \langle M/A * B \rangle : \Gamma \xrightarrow{+} (x \colon A) * (y \colon B) \Rightarrow \Delta$. We take $\rho = (\tau \circ \sigma') \circ \langle M/A * B \rangle$ and $\delta = id$. The assertions then follow by associativity of $\circ$ for $=$ and $\equiv$.
- τ is $\tau'_\bullet \colon (\Delta', x \colon B) \Rightarrow (\Phi', x \colon A)$. Case distinction on the last rule of the derivation of σ:

- σ is $\langle \sigma', M \rangle \colon \Gamma' \xrightarrow{+} \Delta', x \colon B$. We know $\sigma' \colon \Gamma' \Rightarrow \Delta'$. We satisfy 2 by taking $\rho = \langle \tau' \circ \sigma', M \rangle$ and $\delta = id$. We have $\tau \circ \sigma = \tau'_\bullet \circ \langle \sigma', M \rangle = \langle \tau' \circ \sigma', M \rangle = \rho \circ \delta$. Note that each part of this sequence is derivable. Furthermore, the assertion for $\equiv$ follows by Lemma 6.4.10.1 as both σ' and τ' are structural.
- σ is $\sigma'_\bullet \colon (\Gamma', x \colon C) \to (\Delta', x \colon B)$. We apply the induction hypothesis to τ' and σ'. There are two cases to consider. First, we obtain δ' making the following diagram commute.

$$\begin{array}{ccc} \Gamma' & \xrightarrow[\sigma']{+} & \Delta' \\ & {\scriptstyle \delta'} \searrow & \Big\Downarrow {\scriptstyle \tau'} \\ & & \Phi' \end{array}$$

  We can satisfy 1 by letting $\delta = \delta'_\bullet$. Second, we obtain ρ' and δ' such that

$$\begin{array}{ccc} \Gamma' & \xrightarrow[\sigma']{+} & \Delta' \\ \Big\Downarrow {\scriptstyle \delta'} & & \Big\Downarrow {\scriptstyle \tau'} \\ \Theta' & \xrightarrow[\rho']{+} & \Phi' \end{array}$$

  commutes, and $m(\sigma') \geq m(\rho')$ and $\tau' \circ \sigma' \equiv \rho' \circ \delta'$ hold. Take $\rho = \rho'_\bullet$ and $\delta = \delta'_\bullet$. We have $\tau \circ \sigma = \tau'_\bullet \circ \sigma'_\bullet = (\tau' \circ \sigma')_\bullet = (\rho' \circ \delta')_\bullet = \rho'_\bullet \circ \delta'_\bullet = \rho \circ \delta$. The assertion about $\equiv$ is trivial as $\tau'_\bullet \circ \sigma'_\bullet$ and $(\tau' \circ \sigma')_\bullet$ have the same translation. Finally, we have $m(\sigma) = m(\sigma') \geq m(\rho') = m(\rho)$.
- σ is $\sigma' * id \colon \Gamma' * \Phi \xrightarrow{+} \Delta' * \Phi$ or $id * \sigma' \colon \Phi * \Gamma' \xrightarrow{+} \Phi * \Delta'$. Impossible because of the form of the codomain.
- The case where σ is $\sigma' \circ \langle M / A * B \rangle$ is as above.

- τ is w. Case distinction on the last rule of the derivation of σ:
  - $\langle \sigma', M \rangle \colon \Gamma' \xrightarrow{+} \Delta', x \colon B$. We satisfy 1 by letting $\delta = \sigma'$. We have $\mathrm{w} \circ \langle \sigma', M \rangle \equiv \mathrm{w} \circ \sigma'_\bullet \circ \langle M \rangle \equiv \sigma' \circ \mathrm{w} \circ \langle M \rangle \equiv \sigma' \circ id \equiv \sigma'$.
  - σ is $\sigma'_\bullet \colon \Gamma', x \colon C \to \Delta', x \colon B$. We satisfy 2 by letting $\rho = \sigma'$ and $\delta = \mathrm{w}$ as in

$$\begin{array}{ccc} \Gamma', x \colon C & \xrightarrow[\sigma'_\bullet]{+} & \Delta', x \colon B \\ \Big\Downarrow {\scriptstyle \mathrm{w}} & & \Big\Downarrow {\scriptstyle \mathrm{w}} \\ \Gamma' & \xrightarrow[\sigma']{+} & \Delta' \end{array}$$

    Thus, $\tau \circ \sigma = \mathrm{w} \circ \sigma'_\bullet = \sigma' \circ \mathrm{w} = \rho \circ \delta$. Moreover, $\mathrm{w} \circ \sigma'_\bullet \equiv \sigma' \circ \mathrm{w}$ as, given $\sigma' \Downarrow \chi'$, both sides translate to $() \circ \chi'$ and $\chi' \circ ()$ respectively. Finally, $m(\sigma) = m(\sigma') = m(\rho)$.
  - σ is $\sigma' * id \colon \Gamma' * \Phi \xrightarrow{+} \Delta' * \Phi$ or $id * \sigma' \colon \Phi * \Gamma' \xrightarrow{+} \Phi * \Delta'$. Impossible because of the form of the codomain.

  - The case where σ is $\sigma' \circ \langle M/A * B \rangle$ is as above.

- τ is unit. We have $\sigma : \Gamma \xrightarrow{*} \Delta$. Satisfy 2 by taking $\rho = \sigma * \diamond$ and $\delta = \text{unit}$.

- τ is unit^{-1}. Case distinction on the last rule of the derivation of σ:

  - σ is $\langle \sigma', M \rangle : \Gamma' \xrightarrow{+} \Delta', x : B$. Impossible because of the form of the codomain.
  - σ is $\sigma'_\bullet : (\Gamma', x : C) \to (\Delta', x : B)$. Impossible because of the form of the codomain.
  - $(\sigma' * id) : \Gamma' * \diamond \to \Delta' * \diamond$. We satisfy 2 by letting $\rho = \sigma'$ and $\delta = \text{unit}^{-1}$. With this choice we have $\tau \circ \sigma = \text{unit}^{-1} \circ (\sigma' * id) = \sigma' \circ \text{unit}^{-1} = \rho \circ \delta$. For $\equiv$, we observe that, for $\sigma' \Downarrow \chi'$, $\tau \circ \sigma$ and $\rho \circ \delta$ translate to $() \circ (\chi' \circ ())$ and $\chi' \circ ()$.
  - $(id * \sigma')$. We satisfy 1 by letting $\delta = \text{unit}^{-1} \circ (id * !)$. We have $\tau \circ \sigma = \text{unit}^{-1} \circ (id * \sigma') = \text{unit}^{-1} \circ (id * !) = \delta$. The assertion about $\equiv$ follows by observing that for any two $\sigma, \tau : \Gamma \to \diamond$ it holds that $\sigma \equiv \tau$.

- τ is swap. Case distinction on σ:

  - $\langle \sigma', M \rangle : \Gamma \xrightarrow{+} \Delta, x : A$. Impossible because of the form of the codomain.
  - σ is $\sigma'_\bullet : \Gamma', x : C \to \Delta', x : B$. Impossible because of the form of the codomain.
  - $(\sigma' * id)$. Take $\rho = (id * \sigma')$ and $\delta = \text{swap}$.

- τ is assoc or assoc^{-1}. Similar to swap.

□

Lemma 6.5.11. *For all $\sigma : \Gamma \xrightarrow{*} \Delta$ and all $\tau : \Delta \xrightarrow{*} \Phi$ there exists a derivation of some $\rho : \Gamma \xrightarrow{*} \Phi$, such that $\tau \circ \sigma = \rho : \Gamma \to \Phi$, $m(\sigma) \geq m(\rho)$ and $\tau \circ \sigma \equiv \rho : \Gamma \to \Phi$ all hold.*

Proof. Straightforward induction on the derivation of σ, using Lemma 6.5.10, as each map $\xrightarrow{*}$ may be seen as being of the form $\Rightarrow \xrightarrow{+} \ldots \xrightarrow{+}$ for zero or more $\xrightarrow{+}$s. □

The following three lemmas all follow by straightforward induction on the derivation of σ.

Lemma 6.5.12. *For all $\sigma : \Gamma \xrightarrow{*} \Delta$, $\Delta \vdash A$ Type, $\Gamma \vdash A[\sigma] = B$ Type and $x \notin \Gamma, \Delta$, there exists a derivation of some $\rho : \Gamma, x : B \xrightarrow{*} \Delta, x : A$, such that $\sigma_\bullet = \rho : \Gamma, x : B \to \Delta, x : A$, $m(\sigma_\bullet) \geq m(\rho)$ and $\sigma_\bullet \equiv \rho : \Gamma, x : B \to \Delta, x : A$ all hold.*

Lemma 6.5.13. *For all $\sigma : \Gamma \xrightarrow{*} \Delta$ and $\vdash \Phi$ Bunch with $\Phi \cap (\Gamma \cup \Delta) = \emptyset$, there exists a derivation of some $\rho : \Gamma * \Phi \xrightarrow{*} \Delta * \Phi$, such that $\sigma * id = \rho : \Gamma * \Phi \to \Delta * \Phi$, $m(\sigma * id) \geq m(\rho)$ and $\sigma * id \equiv \rho : \Gamma * \Phi \to \Delta * \Phi$ all hold.*

Lemma 6.5.14. *For all $\sigma : \Gamma \xrightarrow{*} \Delta$ and $\vdash \Phi$ Bunch with $\Phi \cap (\Gamma \cup \Delta) = \emptyset$, there exists a derivation of some $\rho : \Phi * \Gamma \xrightarrow{*} \Phi * \Delta$, such that $id * \sigma = \rho : \Phi * \Gamma \to \Phi * \Delta$, $m(id * \sigma) \geq m(\rho)$ and $id * \sigma \equiv \rho : \Phi * \Gamma \to \Phi * \Delta$ all hold.*

Lemma 6.5.15. *For all $\sigma\colon \Gamma \to \Delta$ there exists a derivation of some $\rho\colon \Gamma \xrightarrow{*} \Delta$, such that $\sigma = \rho\colon \Gamma \to \Phi$, $m(\sigma) \geq m(\rho)$ and $\sigma \equiv \rho\colon \Gamma \to \Delta$ all hold.*

Proof. By induction on the derivation of $\sigma\colon \Gamma \to \Delta$. Case distinction on the last rule:

- ($\to$-BU-CONV). Apply Lemma 6.5.9.2.
- (SUB-STR). Apply (SUB*-STR).
- (SUB-COMP). Apply induction hypothesis and Lemma 6.5.11.
- (SUB-$*$). The last rule is:

$$(\text{SUB-}*)\ \frac{\sigma \Downarrow \chi\colon \Gamma \to \Delta \qquad \tau \Downarrow \xi\colon \Phi \to \Psi}{\sigma * \tau \Downarrow \chi\{\xi/\Psi\}\colon \Gamma * \Phi \to \Delta * \Psi}\ \emptyset = (\Gamma \cup \Delta) \cap (\Phi \cup \Psi)$$

  By induction hypothesis, we have $\sigma'\colon \Gamma \xrightarrow{*} \Delta$ and $\tau'\colon \Phi \xrightarrow{*} \Psi$ with $\sigma = \sigma'$, $\tau = \tau'$, $\sigma \equiv \sigma'$, $\tau \equiv \tau'$, $m(\sigma) \geq m(\sigma')$ and $m(\tau) \geq m(\tau')$. By the side-condition of (SUB-$*$), we can use Lemmas 6.5.13 and 6.5.14 and combine the results using (SUB*-COMP). It follows from the definition of the measure m and Lemma 6.4.10.8 that the combined map enjoys the required properties.

- (SUB-LIFT). Apply induction hypothesis and Lemma 6.5.12.
- (SUB-SUB). Apply (SUB$^+$-SUB) and precompose with *id* to get a $\xrightarrow{*}$-map.
- (SUB-LET). Apply (SUB$^+$-LET) and precompose with *id* to get a $\xrightarrow{*}$-map.

□

Having shown that substitutions can be brought into a suitable form, we now give a series of inversion lemmas that will be used in the proof of the Main Lemma. The following series of lemmas provides inversion principles for types and terms under substitution. In addition to the normal inversion, these lemmas also show that if two types/terms have the same syntactic translation then so do the subterms obtained by inversion. For example, if $(\lambda A.M)[\sigma]$ and $(\lambda B.N)[\tau]$ are derivable terms with the same translation, then so are $M[\sigma_\bullet]$ and $N[\tau_\bullet]$.

First we state a lemma that allows us to restrict the number of cases we have to consider.

Lemma 6.5.16.

1. *If $\Gamma \vdash A[\sigma] \Downarrow A'$* Type *is derivable and A is not of the form $B[\tau]$, then there exists a derivation of the same judgement that ends with an application of one of the rules* (C-TY)*,* ($*$-TY)*,* (Π-TY) *and* (Π^*-TY) *followed by a single application of* (SUBST).
2. *If $\Gamma \vdash M[\sigma] \Downarrow M' : A$ is derivable and M is not of the form $N[\tau]$, then there exists a derivation of the same judgement that ends with an application of* (PROJ) *or an introduction rule or elimination rule, followed by an application of* (SUBST) *and followed by an application of* (TY-CONV).

Proof.

1. Examining the rules, a derivation of $\Gamma \vdash A[\sigma] \Downarrow A'$ Type must end in one of the rules (C-TY), ($*$-TY), (Π-TY) and (Π^*-TY), followed by zero or more applications of (BU-CONV), followed by (SUBST), followed by zero or more applications of (BU-CONV). We consider the case where (Π-TY) has been used. The other cases are similar. Since bunch conversion is reflexive and transitive, we can assume that before and after (SUBST) the rule (BU-CONV) is applied exactly once. The end of the derivation thus has the following form.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \Pi A.B \Downarrow \Pi x\colon C.D \text{ Type}}(\Pi\text{-TY}) \qquad \vdash \Gamma = \Delta \text{ Bunch}}{\Delta \vdash \Pi A.B \Downarrow \Pi x\colon C.D \text{ Type}}(\text{BU-CONV}) \qquad \sigma \Downarrow \chi\colon \Phi \to \Delta}{\Phi \vdash (\Pi A.B)[\sigma] \Downarrow (\Pi x\colon C.D)\ \chi \text{ Type}}(\text{SUBST}) \qquad \vdash \Phi = \Psi \text{ Bunch}}{\Psi \vdash (\Pi A.B)[\sigma] \Downarrow (\Pi x\colon C.D)\ \chi \text{ Type}}(\text{BU-CONV})$$

   Then the following derivation has the required form.

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \Pi A.B \Downarrow \Pi x\colon C.D \text{ Type}}(\Pi\text{-TY}) \qquad \dfrac{\dfrac{\vdash \Phi = \Psi \text{ Bunch}}{\vdash \Psi = \Phi \text{ Bunch}} \qquad \sigma\colon \Phi \to \Delta \qquad \dfrac{\vdash \Gamma = \Delta \text{ Bunch}}{\vdash \Delta = \Gamma \text{ Bunch}}}{\sigma \Downarrow \chi\colon \Psi \to \Gamma}}{\Psi \vdash (\Pi A.B)[\sigma] \Downarrow (\Pi x\colon C.D)\ \chi \text{ Type}}(\text{SUBST})$$

2. First, it is easily seen that (BU-CONV) commutes with (TY-CONV). Second, we note that (SUBST) after (TY-CONV) can be transformed into (TY-CONV) after (SUBST), as $\sigma\colon \Gamma \to \Delta$ and $\Delta \vdash A = B$ Type imply $\Gamma \vdash A[\sigma] = B[\sigma]$ Type. Like in the first point, we can assume that zero or more applications of (BU-CONV) can be made into exactly one application of (BU-CONV). We can therefore assume the derivation to end with an introduction or elimination followed by a single application of (BU-CONV), followed by (SUBST), followed by (BU-CONV) and finally followed by (TY-CONV). The applications of (BU-CONV) can be absorbed by the substitution, as in the first case above.

□

Lemma 6.5.17. *If* $\Gamma \vdash (A_1 * A_2)[\sigma] \Downarrow C$ Type *and* $\Gamma \vdash (B_1 * B_2)[\tau] \Downarrow C$ Type *then there exist derivable sequents* $\vdash A_1 \Downarrow C_1$ Type, $\vdash B_1 \Downarrow C_1$ Type, $\vdash A_2 \Downarrow C_2$ Type *and* $\vdash B_2 \Downarrow C_2$ Type.

Proof. By Lemma 6.5.16, each derivation must end in ($*$-TY) followed by (SUBST). Examination of these rule applications gives us derivable judgements:

$$\begin{array}{ll} \sigma \Downarrow \chi\colon \Gamma \to \Delta & \tau \Downarrow \xi\colon \Gamma \to \Phi \\ \vdash A_1 \Downarrow A_1' \text{ Type} & \vdash B_1 \Downarrow B_1' \text{ Type} \\ \vdash A_2 \Downarrow A_2' \text{ Type} & \vdash B_2 \Downarrow B_2' \text{ Type} \\ C = (A_1' * A_2')\ \chi & C = (B_1' * B_2')\ \xi \end{array}$$

Hence, $A_1'\ \chi = B_1'\ \xi$ and $A_2'\ \chi = B_2'\ \xi$ hold. By Lemma 6.4.6, the types A_1', A_2', B_1' and B_2' do not contain free variables, which implies $A_1' = B_1'$ and $A_2' = B_2'$. □

Lemma 6.5.18. *If* $\Gamma \vdash (\Pi A_1.A_2)[\sigma] \Downarrow C$ Type *and* $\Gamma \vdash (\Pi B_1.B_2)[\tau] \Downarrow C$ Type *then there exist derivable sequents* $\Gamma \vdash A_1[\sigma] \Downarrow C_1$ Type, $\Gamma \vdash B_1[\tau] \Downarrow C_1$ Type, $\Gamma, x\colon A_1[\sigma] \vdash A_2[\sigma_\bullet] \Downarrow C_2$ Type *and* $\Gamma, x\colon B_1[\tau] \vdash B_2[\tau_\bullet] \Downarrow C_2$ Type.

The proof is similar to that of the next lemma.

Lemma 6.5.19. *If* $\Gamma \vdash (\Pi^* A_1.A_2)[\sigma] \Downarrow C$ Type *and* $\Gamma \vdash (\Pi^* B_1.B_2)[\tau] \Downarrow C$ Type *then there exist derivable sequents* $\vdash A_1 \Downarrow C_1$ Type, $\vdash B_1 \Downarrow C_1$ Type, $\Gamma * x\colon A_1 \vdash A_2[\sigma * id] \Downarrow C_2$ Type *and* $\Gamma * x\colon B_1 \vdash B_2[\tau * id] \Downarrow C_2$ Type.

Proof. By Lemma 6.5.16, each derivation must end in (Π^*-Ty) followed by (Subst), so that we have:

$$\frac{\sigma \Downarrow \chi\colon \Gamma \to \Delta \qquad \vdash A_1 \Downarrow A_1' \text{ Type} \qquad \Delta * x\colon A_1 \vdash A_2 \Downarrow A_2' \text{ Type} \qquad C = (\Pi^* x\colon A_1'.A_2')\ \chi}{\tau \Downarrow \xi\colon \Gamma \to \Phi \qquad \vdash B_1 \Downarrow B_1' \text{ Type} \qquad \Phi * x\colon B_1 \vdash B_2 \Downarrow B_2' \text{ Type} \qquad C = (\Pi^* x\colon B_1'.B_2')\ \xi}$$

From $(\Pi^* x\colon A_1'.A_2')\ \chi = (\Pi^* x\colon B_1'.B_2')\ \xi$ we obtain $A_1'\ \chi = B_1'\ \xi$ and $A_2'\ \chi\{z/x\} = B_2'\ \xi\{z/x\}$ for some fresh z. Let $C_1 = A_1'$ and $C_2 = A_2'\ \chi\{z/x\}$. By Lemma 6.4.5, we have $\Delta * z\colon A_1 \vdash A_2 \Downarrow A_2'\ \{z/x\}$ Type and $\Phi * z\colon B_1 \vdash B_2 \Downarrow B_2'\ \{z/x\}$ Type. Therefore, we get $\Gamma * z\colon A_1 \vdash A_2[\sigma * id] \Downarrow A_2'\ \{z/x\}\ \chi$ Type and $\Gamma * z\colon B_1 \vdash B_2[\tau * id] \Downarrow B_2'\ \{z/x\}\ \xi$ Type. Since z is fresh, we have $A_2'\ \{z/x\}\ \chi = A_2'\ \chi\{z/x\}$ and $B_2'\ \{z/x\}\ \xi = B_2'\ \xi\{z/x\}$. Using $A_2'\ \chi\{z/x\} = B_2'\ \xi\{z/x\}$, we get that both $A_2[\sigma * id]$ and $B_2[\tau * id]$ translate to C_2. Furthermore, by Lemma 6.4.6, neither A_1' nor B_1' contain free variables, which implies $A_1' = A_1'\ \chi = B_1'\ \xi = B_1'$. Hence, the syntax translation of both A_1 and B_1 is C_1. □

Lemma 6.5.20. *Let* $\delta\colon (x\colon A) * (y\colon B) \Rightarrow (\Delta, z\colon C)$ *be a structural map such either that* $x = z$ *or* $y = z$ *holds. If* $\Gamma \vdash 1[\delta][\langle M/A * B\rangle][\sigma] \Downarrow R : C$ *and* $\Gamma \vdash 1[\delta][\langle N/A * B\rangle][\tau] \Downarrow R : D$ *then there exists* Q *such that* $\Gamma \vdash M[\sigma] \Downarrow Q : A{*}B$ *and* $\Gamma \vdash N[\tau] \Downarrow Q : A{*}B$.

Proof. We spell out the case where $x = z$ holds. Since, by Lemma 6.5.16, there is a derivation of $\Gamma \vdash 1[\delta][\langle M/A * B\rangle][\sigma] \Downarrow R : C$ that ends in (Subst) followed by (Ty-Conv), we have derivations of $\sigma \Downarrow \chi\colon \Gamma \to \Gamma_1$ and $\Gamma_1 \vdash 1[\delta][\langle M/A * B\rangle] \Downarrow R_1 : C'$ and $\Gamma_1 \vdash C = C'[\sigma]$ Type and $R = R_1\ \chi$. Using Lemma 6.5.16 again, and doing analogous steps for the other derivation in the proposition, we get:

$$\frac{\sigma \Downarrow \chi\colon \Gamma \to \Gamma_1 \qquad \Gamma_1 \vdash M \Downarrow M' : A{*}B \Downarrow A'{*}B' \qquad x\colon A * y\colon B \vdash 1[\delta] \Downarrow x : C'' \qquad \Gamma \vdash C = C''[\langle M/A * B\rangle][\sigma] \text{ Type} \qquad R = x\ \{r_{A',B'}(M')/y\}\{l_{A',B'}(M')/x\}\ \chi}{\tau \Downarrow \xi\colon \Gamma \to \Delta_1 \qquad \Delta_1 \vdash N \Downarrow N' : A{*}B \Downarrow A''{*}B'' \qquad x\colon A * y\colon B \vdash 1[\delta] \Downarrow x : D'' \qquad \Gamma \vdash D = D''[\langle M/A * B\rangle][\tau] \text{ Type} \qquad R = x\ \{r_{A'',B''}(N')/y\}\{l_{A'',B''}(N')/x\}\ \xi}$$

Using (SUBST), we can therefore derive

$$\Gamma \vdash M[\sigma] \Downarrow M'\ \chi : (A*B)[\sigma] = (A*B), \qquad \Gamma \vdash N[\tau] \Downarrow N'\ \xi : (A*B)[\tau] = (A*B).$$

It remains to show that $M'\ \chi = N'\ \xi$ holds. This can be seen because we have $l_{A'\ \chi,B'\ \chi}(M'\ \chi) = x\,\{r_{A',B'}(M')/y\}\{l_{A',B'}(M')/x\}\ \chi = R = x\,\{r_{A'',B''}(N')/y\}\{l_{A'',B''}(N')/x\}\ \xi = l_{A''\ \xi,B''\ \xi}(N'\ \xi)$. □

Lemma 6.5.21. *If* $\Gamma \vdash (\lambda A.M)[\sigma] \Downarrow R : C$ *and* $\Gamma \vdash (\lambda B.N)[\tau] \Downarrow R : D$ *then there exist E, F, S, T and x such that* $\Gamma \vdash A[\sigma] \Downarrow S$ Type *and* $\Gamma \vdash B[\tau] \Downarrow S$ Type *and* $\Gamma \vdash C = \Pi A[\sigma].E[\sigma_\bullet]$ Type *and* $\Gamma \vdash D = \Pi B[\tau].F[\tau_\bullet]$ Type *and* $\Gamma, x\colon A[\sigma] \vdash M[\sigma_\bullet] \Downarrow T : E[\sigma_\bullet]$ *and* $\Gamma, x\colon B[\tau] \vdash N[\tau_\bullet] \Downarrow T : F[\tau_\bullet]$ *are all derivable.*

Proof. Using Lemma 6.5.16 as in the previous lemma, we get derivations of the following judgements.

$$\sigma \Downarrow\ \chi : \Gamma \to \Delta \qquad \tau \Downarrow\ \xi : \Gamma \to \Phi$$
$$\Delta \vdash A \Downarrow A_s \text{ Type} \qquad \Phi \vdash B \Downarrow B_s \text{ Type}$$
$$\Delta, x\colon A \vdash M \Downarrow M_s : E \qquad \Phi, x\colon B \vdash N \Downarrow N_s : F$$
$$\Gamma \vdash C = (\Pi A.E)[\sigma] \text{ Type} \qquad \Gamma \vdash D = (\Pi B.F)[\tau] \text{ Type}$$
$$R = (\lambda x\colon A_s.M_s)\ \chi \qquad R = (\lambda x\colon B_s.N_s)\ \xi$$

By $(\lambda x\colon A_s.M_s)\ \chi = R = (\lambda x\colon B_s.N_s)\ \xi$, we have $A_s\ \chi = B_s\ \xi$ and $M_s\ \chi\{z/x\} = N_s\ \xi\{z/x\}$ for some fresh variable z. Using Lemma 6.4.5, we get

$$\Delta, z\colon A \vdash M \Downarrow M_s\ \{z/x\} : E \qquad \Phi, z\colon B \vdash N \Downarrow N_s\ \{z/x\} : F$$

Using (SUBST), we can derive

$$\Gamma \vdash A[\sigma] \Downarrow A_s\ \chi \text{ Type} \qquad \Gamma \vdash B[\tau] \Downarrow B_s\ \xi \text{ Type}$$
$$\Gamma, z\colon A[\sigma] \vdash M[\sigma_\bullet] \Downarrow M_s\ \{z/x\}\ \chi : E[\sigma_\bullet] \qquad \Gamma, z\colon B[\tau] \vdash N[\tau_\bullet] \Downarrow N_s\ \{z/x\}\ \xi : F[\tau_\bullet]$$

Since z is fresh and x does not occur in the codomain of σ and τ, we have $M_s\ \{z/x\}\ \chi = M_s\ \chi\{z/x\}$ and $N_s\ \{z/x\}\ \xi = N_s\ \xi\{z/x\}$. Therefore, letting $S = A_s\ \chi$ and $T = M_s\ \chi\{z/x\}$, this completes the proof. □

The following lemmas are proved similarly.

Lemma 6.5.22. *If* $\Gamma \vdash (\mathrm{app}_{[A_1]A_2}(M_1,M_2))[\sigma] \Downarrow R : C$ *and* $\Gamma \vdash (\mathrm{app}_{[B_1]B_2}(N_1,N_2))[\tau] \Downarrow R : D$ *then there exist S, T, U, V and x such that* $\Gamma \vdash A_1[\sigma] \Downarrow S$ Type *and* $\Gamma \vdash B_1[\tau] \Downarrow S$ Type *and* $\Gamma, x\colon A_1[\sigma] \vdash A_2[\sigma_\bullet] \Downarrow T$ Type *and* $\Gamma, x\colon B_1[\tau] \vdash B_2[\tau_\bullet] \Downarrow T$ Type *and* $\Gamma \vdash M_1[\sigma] \Downarrow U : (\Pi A_1.A_2)[\sigma]$ *and* $\Gamma \vdash M_2[\sigma] \Downarrow V : A_1[\sigma]$ *and* $\Gamma \vdash N_1[\tau] \Downarrow U : (\Pi B_1.B_2)[\tau]$ *and* $\Gamma \vdash N_2[\tau] \Downarrow V : B_1[\tau]$ *are all derivable.*

Lemma 6.5.23. *If* $\Gamma \vdash (\lambda^* A.M)[\sigma] \Downarrow R : C$ *and* $\Gamma \vdash (\lambda^* B.N)[\tau] \Downarrow R : D$ *then there exist E, F, S, T and x such that* $\vdash A \Downarrow S$ Type *and* $\vdash B \Downarrow S$ Type *and* $\Gamma \vdash C = \Pi^* A.E[\sigma * id]$ Type *and* $\Gamma \vdash D = \Pi^* B.F[\tau * id]$ Type *and* $\Gamma * x\colon A \vdash M[\sigma * id] \Downarrow T : E[\sigma * id]$ *and* $\Gamma * x\colon B \vdash N[\tau * id] \Downarrow T : F[\tau * id]$ *are all derivable.*

Lemma 6.5.24. *If* $\Gamma \vdash (\mathrm{app}^*_{[A_1]A_2}(M_1,M_2))[\sigma] \Downarrow R : C$ *and* $\Gamma \vdash (\mathrm{app}^*_{[B_1]B_2}(N_1,N_2))[\tau] \Downarrow R : D$ *then there exist S, T, U, V and x such that* $\vdash A_1 \Downarrow S$ Type *and* $\vdash B_1 \Downarrow S$ Type *and* $\Gamma \vdash A_2[id * \langle !, M_2\rangle][\sigma] \Downarrow T$ Type *and* $\Gamma \vdash B_2[id * \langle !, N_2\rangle][\tau] \Downarrow T$ Type *and* $\Gamma \vdash M_1[\dot{\pi}_1 \circ \sigma] \Downarrow U : (\Pi^* A_1.A_2)[\dot{\pi}_1 \circ \sigma]$ *and* $\Gamma \vdash M_2[\dot{\pi}_2 \circ \sigma] \Downarrow V : A_1[!_\Gamma]$ *and* $\Gamma \vdash N_1[\dot{\pi}_1 \circ \tau] \Downarrow U : (\Pi^* B_1.B_2)[\dot{\pi}_1 \circ \tau]$ *and* $\Gamma \vdash N_2[\dot{\pi}_2 \circ \tau] \Downarrow V : B_1[!_\Gamma]$ *are all derivable.*

Lemma 6.5.25. *If* $\Gamma \vdash (\mathrm{pair}^*_{A_1,A_2}(M_1,M_2))[\sigma] \Downarrow R : A$, *and* $\Gamma \vdash (\mathrm{pair}^*_{B_1,B_2}(N_1,N_2))[\tau] \Downarrow R : B$ *then there exist S, T, A′, B′ and types* $\vdash A_1 \Downarrow A'$ Type, $\vdash A_2 \Downarrow A'$ Type, $\vdash B_1 \Downarrow B'$ Type *and* $\vdash B_2 \Downarrow B'$ Type, *such that* $\Gamma \vdash A_1 * A_2 = A$ Type *and* $\Gamma \vdash B_1 * B_2 = B$ Type *and* $\Gamma \vdash M_1[\dot{\pi}_1 \circ \sigma] \Downarrow S : A_1[!_\Gamma]$ *and* $\Gamma \vdash N_1[\dot{\pi}_1 \circ \tau] \Downarrow S : B_1[!_\Gamma]$ *and* $\Gamma \vdash M_2[\dot{\pi}_2 \circ \sigma] \Downarrow T : A_2[!_\Gamma]$ *and* $\Gamma \vdash N_2[\dot{\pi}_2 \circ \tau] \Downarrow T : B_2[!_\Gamma]$ *are all derivable.*

Proof. Using Lemma 6.5.16, we have $\Gamma \vdash A_1 * A_2 = A$ Type, $\Gamma \vdash B_1 * B_2 = B$ Type, $\sigma\colon \Gamma \to \Delta_1 * \Delta_2$ and $\tau\colon \Gamma \to \Phi_1 * \Phi_2$, as well as, for all $i \in \{1,2\}$, $\vdash A_i \Downarrow A'_i$ Type, $\vdash B_i \Downarrow B'_i$ Type, $\Delta_i \vdash M_i \Downarrow S_i : A_i[!_{\Delta_i}]$ and $\Phi_i \vdash N_i \Downarrow T_i : B_i[!_{\Phi_i}]$. Moreover, R has the form $\mathrm{pair}^*_{A'_1,A'_2}(S_1,S_1)$ and also the form $\mathrm{pair}^*_{B'_1,B'_2}(T_1,T_2)$. Hence, $A'_i = B'_i$ and $S_i = T_i$ hold for all $i \in \{1,2\}$. The derivation of the required statements now follows using substitution. □

Lemma 6.5.26 (Main Lemma)**.**

1. *If* $\Gamma \vdash A \Downarrow C$ Type *and* $\Gamma \vdash B \Downarrow C$ Type *then* $\Gamma \vdash A = B$ Type.
2. *If* $\Gamma \vdash M \Downarrow R : A$ *and* $\Gamma \vdash N \Downarrow R : B$ *then* $\Gamma \vdash A = B$ Type *and* $\Gamma \vdash M = N : A$.

Proof. We establish both 1 and 2 simultaneously by induction on $m(A) + m(B)$ and $m(M) + m(N)$ respectively.

1. In general, A has the form $A'[\sigma_1]\ldots[\sigma_n]$ for $n \geq 0$. We have $\Gamma \vdash A = A'[id \circ \sigma_1 \circ \cdots \circ \sigma_n]$ Type, where the right hand term does not have larger measure. Hence, we can assume that A is $A'[\sigma]$ such that A' does not have the form $A''[\sigma']$. We make the same assumption for B. Case distinction on the type A'.

   - A is $T(\sigma)$. We can assume A to be of this form since we have $T(\tau)[\sigma] = T(\tau \circ \sigma)$. We know that C has the form $T(R_1,\ldots,R_n)$. Since both $A \Downarrow C$ and $B \Downarrow C$ hold, the type B must have the form $T(\tau)$. Since $T \in \mathscr{T}(\Delta)$ for a unique pre-context Δ, we have $\sigma\colon \Gamma \to \Delta$ and $\tau\colon \Gamma \to \Delta$. For each structural map $\delta\colon \Delta \Rightarrow (\Phi, x\colon D)$, we can derive $\Gamma \vdash 1[\delta][\sigma] \Downarrow R_i : D[\mathrm{w}][\delta][\sigma]$ and $\Gamma \vdash 1[\delta][\tau] \Downarrow R_i : D[\mathrm{w}][\delta][\tau]$ for some $i \in \{1,\ldots,n\}$. Because we have $m(T(\sigma)) = 2 + m(\sigma) > 1 + m(\sigma) = m(1[\delta][\sigma])$, we can use the induction hypothesis on terms to get $\Gamma \vdash 1[\delta][\sigma] = 1[\delta][\tau] : D[\mathrm{w}][\delta][\sigma] = D[\mathrm{w}][\delta][\tau]$. Since δ was arbitrary, we can use Lemma 6.5.4 to obtain $\sigma = \tau\colon \Gamma \to \Delta$. The case is completed by using (C-Ty-Cgr) to derive $\Gamma \vdash A = T(\sigma) = T(\tau) = B$ Type.
   - A is $(A_1 * A_2)[\sigma]$. Then C has $*$ as its outermost constructor, which implies that B must be $(B_1 * B_2)[\tau]$. By Lemma 6.5.17, we have derivations $\vdash A_1 \Downarrow C_1$ Type, $\vdash B_1 \Downarrow C_1$ Type, $\vdash A_2 \Downarrow C_2$ Type and $\vdash B_2 \Downarrow C_2$ Type. From the induction hypothesis, we obtain derivations

of $\vdash A_1 = B_1$ Type and $\vdash A_2 = B_2$ Type. From this we derive $\vdash A_1 * A_2 = B_1 * B_2$ Type. By rule (SUBST) we obtain $\vdash (A_1 * A_2)[!_\Gamma] = (B_1 * B_2)[!_\Gamma]$ Type. Using the substitution equation ($*$-SUB), this gives $\Gamma \vdash (A_1 * A_2)[!_\Gamma] = (A_1 * A_2) = (A_1 * A_2)[\sigma]$ Type. With the corresponding equation for $(B_1 * B_2)$, we have thus derived the required $\Gamma \vdash (A_1 * A_2)[\sigma] = (B_1 * B_2)[\tau]$ Type.

- A is $(\Pi A_1.A_2)[\sigma]$. Then C has a Π as its outermost constructor, which implies that B must be $(\Pi B_1.B_2)[\tau]$. By Lemma 6.5.18 we have $\Gamma \vdash A_1[\sigma] \Downarrow C_1$ Type, $\Gamma \vdash B_1[\tau] \Downarrow C_1$ Type, $\Gamma, x\colon A_1[\sigma] \vdash A_2[\sigma_\bullet] \Downarrow C_2$ Type and $\Gamma, x\colon B_1[\tau] \vdash B_2[\tau_\bullet] \Downarrow C_2$ Type. The induction hypothesis yields $\Gamma \vdash A_1[\sigma] = B_1[\tau]$ Type. Using this, bunch conversion, and the induction hypothesis, we obtain $\Gamma, x\colon A_1[\sigma] \vdash A_2[\sigma_\bullet] = B_2[\tau_\bullet]$ Type. By (Π-TY-CGR), we have $\Gamma \vdash \Pi A_1[\sigma].A_2[\sigma_\bullet] = \Pi B_1[\tau].B_2[\tau_\bullet]$ Type. Using (Π-SUB), we therefore get the required

$$\Gamma \vdash A = (\Pi A_1.A_2)[\sigma] = \Pi A_1[\sigma].A_2[\sigma_\bullet] = \Pi B_1[\tau].B_2[\tau_\bullet] = (\Pi B_1.B_2)[\tau] = B \text{ Type}.$$

- A is $(\Pi^* A_1.A_2)[\sigma]$. Then C has a Π^* as its outermost type constructor, which implies that B must be $(\Pi^* B_1.B_2)[\tau]$. Lemma 6.5.19 yields $\vdash A_1 \Downarrow C_1$ Type, $\vdash B_1 \Downarrow C_1$ Type, $\Gamma * x\colon A_1 \vdash A_2[\sigma * id] \Downarrow C_2$ Type and $\Gamma * x\colon B_1 \vdash B_2[\tau * id] \Downarrow C_2$ Type. The induction hypothesis provides $\vdash A_1 = B_1$ Type. Using this, bunch conversion, and the induction hypothesis, we obtain $\Gamma * x\colon A_1 \vdash A_2[\sigma * id] = B_2[\tau * id]$ Type. By (Π^*-TY-CGR), we have $\Gamma \vdash \Pi^* A_1.A_2[\sigma * id] = \Pi^* B_1.B_2[\tau * id]$ Type. Using (Π^*-SUB) we get the required

$$\Gamma \vdash A = (\Pi^* A_1.A_2)[\sigma] = \Pi^* A_1.A_2[\sigma * id] = \Pi^* B_1.B_2[\tau * id] = (\Pi^* B_1.B_2)[\tau] = B \text{ Type}.$$

2. In general, M has the form $M'[\sigma_1]\ldots[\sigma_n]$ for $n \geq 0$. We have $M = M'[id \circ \sigma_1 \circ \cdots \circ \sigma_n]$, where the right hand term does not have larger measure. Hence, we can assume M to have the form $M'[\sigma]$, where M' does not have the form $M''[\sigma']$. We make a corresponding assumption for N. Moreover, if N has the form $1[\tau]$ then we exchange M and N, so that we can assume that N does not have the form $1[\tau]$ unless M also has the form $1[\sigma]$. We proceed by case distinction on M.

   - M is $1[\sigma]$. We have $\Gamma \vdash 1[\sigma] \Downarrow R : A$. Using Lemma 6.5.16, $\sigma\colon \Gamma \to \Delta, x\colon A_0$ and $\Gamma \vdash A = A_0[\mathrm{w}][\sigma]$ Type are derivable. By Lemma 6.5.15, there exists $\tau : \Gamma \xrightarrow{*} \Delta, x\colon A_0$ such that $\Gamma \vdash 1[\tau] \Downarrow R : A$ and $\Gamma \vdash 1[\sigma] = 1[\tau] : A$ are derivable, and that $m(1[\sigma]) \geq m(1[\tau])$ holds. We continue by case distinction on τ.

     - τ is structural. In this case R is a variable, say x. In order for N to translate to a variable, it must be of the form $1[\rho]$. We can assume $\rho\colon \Gamma \xrightarrow{*} \Phi, y\colon B_0$. If ρ is structural then y must be x, and we can use Lemma 6.5.1 to derive $\Gamma \vdash M = N : A_0 = B_0$, as required. If ρ has a substitution or a let as its main part then we can consider the symmetric case with ρ and τ exchanged, so this case is handled by the following two cases.
     - τ is $\tau'' \circ \tau' \Downarrow \chi'' \circ \chi'$, where the main part of τ'' is a substitution $\langle \delta, M' \rangle$, i.e. χ'' is $\{M''/y\}$. By the form of the codomain of τ, the map τ'' must be either (i) ${\tau''_1}_\bullet$, in which case $y \neq x$, or (ii) $\langle \delta, M' \rangle$, in which case $y = x$.

In case (i), we can assume that the domain of τ'' has the form $\Delta', x\colon A_0[\tau'']$. Hence, we have $\Gamma \vdash 1[\tau'] \Downarrow x\ \chi' : A$ as well as $\Gamma \vdash M = 1[\tau] = 1[\tau''_1{}_\bullet][\tau'] = 1[\tau'] : A$. Since $R = x\ \chi = x\ \{M''/y\}\ \chi' = x\ \chi'$ and $m(M) \geq m(1[\tau]) = m(1[\tau''][\tau']) > m(1[\tau'])$, we can apply the induction hypothesis to get $\Gamma \vdash 1[\tau'] = N : A = B$, which implies $\Gamma \vdash M = N : A = B$, as required.

In case (ii), we have $M[\tau] = 1[\langle \delta, M' \rangle \circ \tau'] = M'[\tau']$. The right-hand term has a strictly smaller measure, so that we can apply the induction hypothesis.

– τ is $\tau'' \circ \tau' \Downarrow \chi'' \circ \chi'$, where the main part of τ'' is a let. In this case, χ'' must be $\{r_{D',E'}(R')/v\}\{l_{D',E'}(R')/u\}$ for some R', u and v, and the let-free term R is of the form $x\{r_{D',E'}(R')/v\}\{l_{D',E'}(R')/u\}\ \chi'$ for some variable x. By the form of its codomain, τ'' must be either (i) $\tau''_1{}_\bullet$, in which case both $x \neq u$ and $x \neq v$ hold; or (ii) $\delta \circ \langle M'/D * E \rangle$, in which case we have $x = u$ or $x = v$. The case (i) is handled as in the above case where the main part of τ'' is a substitution.

It remains to consider case (ii). Since R is either a variable or of the form $l_{D',E'}(R')$ or $r_{D',E'}(R')$, it follows that N must have the form $1[\rho]$ where $\rho\colon \Gamma \xrightarrow{*} (\Phi, y\colon B_0)$. We continue by case distinction on ρ. There are three possible cases. If ρ is structural then R must be the variable y. But by assumption of (ii), we know that R is either $l_{D',E'}(R')$ or $r_{D',E'}(R')$. Hence, we have a contradiction so that the case where ρ is structural cannot occur. If ρ is $\rho'' \circ \rho'$, where ρ'' has a substitution as its main formula, then we can use the induction hypothesis as above. The remaining case is where ρ is $\rho'' \circ \rho' \Downarrow \xi'' \circ \xi'$ and ρ'' has a let as its main formula. Then ξ'' must be $\{r_{D'',E''}(R'')/v'\}\{l_{D'',E''}(R'')/u'\}$. As in the case for τ'', there are two possible cases for ρ'', analogous to (i) and (ii). The first case is handled by the induction hypothesis just as the case for (i) above. In the second case ρ'' is $\delta' \circ \langle N'/F * G \rangle$ and either $y = u'$ or $y = v'$ holds. Without loss of generality, we may assume $y = u'$. We can assume the choice of fresh variables to be such that $u' = u$ and $v' = v$. Since the outermost term in R is either l or r, this implies $x = y$. Let $\delta''\colon (u\colon F) * (v\colon G) \Rightarrow (u\colon F)$. It follows using (Str-Eq) that $1[\delta''] = 1[\delta']$ holds. Now we have $\vdash D \Downarrow D'$ Type and $\vdash F \Downarrow E'$ Type, since Lemma 6.4.4 gives us $\vdash D \Downarrow D''$ Type and $\vdash E \Downarrow E''$ Type and from $l_{D',E'}(R'')\ \chi' = R = l_{D'',E''}(R'')\ \xi'$ we get $D' = D''$ and $E' = E''$. By induction hypothesis, we get $\vdash D = F$ Type, and similarly $\vdash E = G$ Type. By conversion, we then have $\delta''\colon (u\colon D) * (v\colon E) \Rightarrow (u\colon D)$. Using (Str-Eq), we obtain $1[\delta''] = 1[\delta]$. Using ($*$-E-Cgr) we moreover get $\langle N'/F * G \rangle = \langle N'/D * E \rangle$, so that we have

$$\Gamma \vdash M = 1[\delta''][\langle M'/D * E \rangle][\tau'] : A \qquad \Gamma \vdash N = 1[\delta''][\langle N'/D * E \rangle][\rho'] : B$$

By Lemma 6.5.20, we obtain an term S such that the sequents $\Gamma \vdash M'[\tau'] \Downarrow S : D * E$ and $\Gamma \vdash N'[\rho'] \Downarrow S : D * E$ are derivable. We can apply the induction hypothesis to obtain $\Gamma \vdash M'[\tau'] = N'[\rho'] : D * E$.

The case is completed by noting $M = 1[\delta''][\langle M'/D*E\rangle][\tau'] = 1[\delta''][\langle M'[\tau']/D*E\rangle] = 1[\delta''][\langle N'[\rho']/D*E\rangle] = 1[\delta''][\langle N'/D*E\rangle][\rho'] = N$.

- M is $f(\sigma)$. We can assume M to be of this form since we have $f(\tau)[\sigma] = f(\tau\circ\sigma)$. We know that R has the form $f(R_1,\dots,R_n)$. Since both M and N translate to R, and N can by assumption not be of the form $1[\tau]$, the term N must have the form $f(\tau)$. Since $f \in \mathscr{T}(\Delta;A_0)$ for a unique context Δ and type A_0, we have $\sigma\colon \Gamma\to\Delta$, $\Gamma\vdash A = A_0[\sigma]$ Type, $\tau\colon\Gamma\to\Delta$ and $\Gamma\vdash B = A_0[\tau]$ Type. For each structural map $\delta\colon\Delta\Rightarrow(\Phi, x\colon D)$, we can derive $\Gamma\vdash 1[\delta][\sigma]\Downarrow R_i : D[\mathrm{w}][\delta][\sigma]$ and $\Gamma\vdash 1[\delta][\tau]\Downarrow R_i : D[\mathrm{w}][\delta][\tau]$ for some $i\in\{1,\dots,n\}$. Since $m(f(\sigma)) = 2+m(\sigma) > 1+m(\sigma) = m(1[\delta][\sigma])$, we can use the induction hypothesis to get $\Gamma\vdash 1[\delta][\sigma] = 1[\delta][\tau] : D[\mathrm{w}][\delta][\sigma] = D[\mathrm{w}][\delta][\tau]$. Since δ was arbitrary, we can apply Lemma 6.5.4 to get $\sigma=\tau\colon\Gamma\to\Delta$. The case is completed by using (C-TM-CGR) to derive $\Gamma\vdash M = f(\sigma) = f(\tau) = N : A = A_0[\sigma] = A_0[\tau] = B$.

- M is $(\lambda A_0.M_0)[\sigma]$. Since both $M\Downarrow R$ and $N\Downarrow R$, the term N must have the form $(\lambda B_0.N_0)[\tau]$. By Lemma 6.5.21, we have derivations of $\Gamma\vdash A_0[\sigma]\Downarrow S$ Type and $\Gamma\vdash B_0[\tau]\Downarrow S$ Type and $\Gamma\vdash A = (\Pi A_0.C)[\sigma]$ Type and $\Gamma\vdash B = (\Pi B_0.D)[\tau]$ Type and $\Gamma, x\colon A_0[\sigma]\vdash M_0[\sigma_\bullet]\Downarrow T : C[\sigma_\bullet]$ and $\Gamma, x\colon B_0[\tau]\vdash N_0[\tau_\bullet]\Downarrow T : D[\tau_\bullet]$.

  Now, we have $m((\lambda A_0.M_0)[\sigma]) = 1+m(A_0)+m(M_0)+m(\sigma)$, $m(A_0[\sigma]) = m(A_0)+m(\sigma)$ and $m(M_0[\sigma_\bullet]) = m(M_0)+m(\sigma)$. We can therefore apply the induction hypothesis to obtain $\Gamma\vdash A_0[\sigma] = B_0[\tau]$ Type and, using (BU-CONV) and the induction hypothesis, also $\Gamma, x\colon A_0[\sigma]\vdash M_0[\sigma_\bullet] = N_0[\tau_\bullet] : C[\sigma_\bullet] = D[\tau_\bullet]$.

  By congruence, substitution equations and transitivity, we get the required

$$\Gamma\vdash A = (\Pi A_0.C)[\sigma] = (\Pi A_0[\sigma].C[\sigma_\bullet]) = (\Pi B_0[\tau].D[\tau_\bullet]) = (\Pi B_0.D)[\tau] = B \text{ Type},$$

$$\Gamma\vdash M = (\lambda A_0.M_0)[\sigma] = (\lambda A_0[\sigma].M_0[\sigma_\bullet]) = (\lambda B_0[\tau].N_0[\tau_\bullet]) = (\lambda B_0.N_0)[\tau] = N : A.$$

- M is $(\mathrm{app}_{[A_1]A_2}(M_1,M_2))[\sigma]$. Since both $M\Downarrow R$ and $N\Downarrow R$, the outermost term constructor in R is an application, which implies that the term N must have the form $(\mathrm{app}_{[B_1]B_2}(N_1,N_2))[\tau]$. Making use of Lemma 6.5.16, we obtain $\Gamma\vdash A = A_2[\langle M_2\rangle][\sigma] = A_2[\sigma_\bullet][\langle M_2[\sigma]\rangle]$ Type and $\Gamma\vdash B = B_2[\langle N_2\rangle][\tau] = B_2[\tau_\bullet][\langle N_2[\tau]\rangle]$ Type. By Lemma 6.5.22, there exist derivable judgements of the following form.

$$\begin{array}{cc}
\Gamma\vdash A_1[\sigma]\Downarrow S \text{ Type} & \Gamma\vdash B_1[\tau]\Downarrow S \text{ Type}\\
\Gamma, x\colon A_1[\sigma]\vdash A_2[\sigma_\bullet]\Downarrow T \text{ Type} & \Gamma, x\colon B_1[\tau]\vdash B_2[\tau_\bullet]\Downarrow T \text{ Type}\\
\Gamma\vdash M_1[\sigma]\Downarrow U : (\Pi A_1.A_2)[\sigma] & \Gamma\vdash N_1[\tau]\Downarrow U : (\Pi B_1.B_2)[\tau]\\
\Gamma\vdash M_2[\sigma]\Downarrow V : A_1[\sigma] & \Gamma\vdash N_2[\tau]\Downarrow V : B_1[\tau]
\end{array}$$

  Each of these sequents has a type/term with smaller measure. We can therefore apply the

induction hypothesis in conjunction with (BU-CONV) to obtain:

$$\Gamma \vdash A_1[\sigma] = B_1[\tau] \text{ Type} \qquad \Gamma, x\colon A_1[\sigma] \vdash A_2[\sigma_\bullet] = B_2[\tau_\bullet] \text{ Type}$$
$$\Gamma \vdash M_1[\sigma] = N_1[\tau] : (\Pi A_1.A_2)[\sigma] = \Pi A_1[\sigma].A_2[\sigma_\bullet] \qquad \Gamma \vdash M_2[\sigma] = N_2[\tau] : A_1[\sigma]$$

These are precisely the premises needed for (Π-E-CGR), so that we get

$$\Gamma \vdash \mathrm{app}_{[A_1[\sigma]]A_2[\sigma_\bullet]}(M_1[\sigma], M_2[\sigma]) = \mathrm{app}_{[B_1[\tau]]B_2[\tau_\bullet]}(N_1[\tau], N_2[\tau]) : A_2[\sigma_\bullet][\langle M_1[\sigma]\rangle].$$

By the equations for substitution, we get

$$\Gamma \vdash (\mathrm{app}_{[A_1]A_2}(M_1, M_2))[\sigma] = (\mathrm{app}_{[B_1]B_2}(N_1, N_2))[\tau] : A_2[\sigma_\bullet][\langle M_1[\sigma]\rangle].$$

Finally, for the types we use congruence of substitution to get $\Gamma \vdash A = A_2[\sigma_\bullet][\langle M_2[\sigma]\rangle] = B_2[\tau_\bullet][\langle M_2[\sigma]\rangle] = B_2[\tau_\bullet][\langle N_2[\tau]\rangle] = B$ Type.

- M is $(\lambda^* A_0.M_0)[\sigma]$. Since both M and N translate to R, the term N must have the form $(\lambda^* B_0.N_0)[\tau]$. By Lemma 6.5.23, we have derivations of $\vdash A_0 \Downarrow S$ Type, $\vdash B_0 \Downarrow S$ Type, $\Gamma \vdash A = (\Pi^* A_0.C)[\sigma]$ Type, $\Gamma \vdash B = (\Pi^* B_0.D)[\tau]$ Type, $\Gamma * x\colon A_0 \vdash M_0[\sigma * id] \Downarrow T : C[\sigma * id]$ and $\Gamma * x\colon B_0 \vdash N_0[\tau * id] \Downarrow T : D[\tau * id]$.

  Now, we have $m((\lambda^* A_0.M_0)[\sigma]) = 1 + m(A_0) + m(M_0) + m(\sigma)$ as well as $m(M_0[\sigma * id]) = m(M_0) + m(\sigma)$. We can therefore apply the induction hypothesis to get $\vdash A_0 = B_0$ Type and $\Gamma * x\colon A_0 \vdash M_0[\sigma * id] = N_0[\tau * id] : C[\sigma * id] = D[\tau * id]$.

  By congruence, substitution equations and transitivity, we get the required

$$\Gamma \vdash A = (\Pi^* A_0.C)[\sigma] = (\Pi^* A_0.C[\sigma * id]) = (\Pi^* B_0.D[\tau * id]) = (\Pi^* B_0.D)[\tau] = B \text{ Type},$$

$$\Gamma \vdash M = (\lambda^* A_0.M_0)[\sigma] = (\lambda^* A_0.M_0[\sigma * id]) = (\lambda^* B_0.N_0[\tau * id]) = (\lambda^* B_0.N_0)[\tau] = N : A.$$

- M is $(\mathrm{app}^*_{[A_1]A_2}(M_1, M_2))[\sigma]$. Since both $M \Downarrow R$ and $N \Downarrow R$ hold, the term N must have the form $(\mathrm{app}^*_{[B_1]B_2}(N_1, N_2))[\tau]$. Using Lemmas 6.5.24 and 6.5.16 we obtain the following derivable judgements, in which we write σ' for $(id * \langle !, M_2\rangle) \circ \sigma$ and τ' for $(id * \langle !, N_2\rangle) \circ \tau$.

$$\begin{array}{ll}
\sigma\colon \Gamma \to \Gamma_1 * \Delta_1 & \tau\colon \Gamma \to \Gamma_2 * \Delta_2 \\
\Gamma_1 \vdash M_1 : \Pi^* A_1.A_2 & \Gamma_2 \vdash N_1 : \Pi^* B_1.B_2 \\
\vdash A_1 \Downarrow S \text{ Type} & \vdash B_1 \Downarrow S \text{ Type} \\
\Gamma \vdash A_2[\sigma'] \Downarrow T \text{ Type} & \Gamma \vdash B_2[\tau'] \Downarrow T \text{ Type} \\
\Gamma \vdash M_1[\dot\pi_1 \circ \sigma] \Downarrow U : (\Pi^* A_1.A_2)[\dot\pi_1 \circ \sigma] & \Gamma \vdash N_1[\dot\pi_1 \circ \tau] \Downarrow U : (\Pi^* B_1.B_2)[\dot\pi_1 \circ \tau] \\
\Gamma \vdash M_2[\dot\pi_2 \circ \sigma] \Downarrow V : A_1[!_\Gamma] & \Gamma \vdash N_2[\dot\pi_2 \circ \tau] \Downarrow V : B_1[!_\Gamma]
\end{array}$$

Since each of these sequents has a type/term with smaller measure, we can use the induction hypothesis to obtain derivations of $\vdash A_1 = B_1$ Type, $\Gamma \vdash A_2[\sigma'] = B_2[\tau']$ Type,

$\Gamma \vdash (\Pi^* A_1.A_2)[\dot{\pi}_1 \circ \sigma] = (\Pi^* B_1.B_2)[\dot{\pi}_1 \circ \tau]$ Type, $\Gamma \vdash N_2[\dot{\pi}_2 \circ \sigma] = N_2[\dot{\pi}_2 \circ \tau] : A_1[!_\Gamma]$ and $\Gamma \vdash M_1[\dot{\pi}_1 \circ \sigma] = N_1[\dot{\pi}_1 \circ \tau] : (\Pi^* A_1.A_2)[\dot{\pi}_1 \circ \sigma]$. We notice that we have $\dot{\pi}_1 \circ \sigma' = \dot{\pi}_1 \circ \sigma$ and $\dot{\pi}_2 \circ \sigma' = \langle !, M_2 \rangle \circ \sigma$, and corresponding equations for τ'. This gives us the equations $(\Pi^* A_1.A_2)[\dot{\pi}_1 \circ \sigma'] = (\Pi^* B_1.B_2)[\dot{\pi}_1 \circ \tau']$, $M_1[\dot{\pi}_1 \circ \sigma'] = N_1[\dot{\pi}_1 \circ \tau']$ and $1[\dot{\pi}_2 \circ \sigma'] = 1[\dot{\pi}_2 \circ \tau']$, in which we omit contexts and types for readability. The assertion now follows using the rule (Π^*-E-CGR) and the substitution equation for app^*.

- M is $(\mathrm{pair}^*_{A_1,A_2}(M_1,M_2))[\sigma]$. Because N has the same syntactic translation as M, it must have the form $(\mathrm{pair}^*_{B_1,B_2}(N_1,N_2))[\tau]$. Using Lemma 6.5.25, we obtain types $\vdash A_1 \Downarrow A'$ Type, $\vdash A_2 \Downarrow A'$ Type, $\vdash B_1 \Downarrow B'$ Type and $\vdash B_2 \Downarrow B'$ Type and the following derivable judgements:

$$\begin{array}{ll} \Gamma \vdash A = A_1 * A_2 \text{ Type} & \Gamma \vdash B = B_1 * B_2 \text{ Type} \\ \Gamma \vdash M_1[\dot{\pi}_1 \circ \sigma] \Downarrow S : A_1[!] & \Gamma \vdash N_1[\dot{\pi}_1 \circ \tau] \Downarrow S : B_1[!] \\ \Gamma \vdash M_2[\dot{\pi}_2 \circ \sigma] \Downarrow T : A_2[!] & \Gamma \vdash N_2[\dot{\pi}_2 \circ \tau] \Downarrow T : B_2[!] \end{array}$$

The induction hypothesis yields $\vdash A_1 = A_2$ Type and $\vdash B_1 = B_2$ Type. This implies $\Gamma \vdash A_1 * A_2 = B_1 * B_2$ Type. Now note that $m(M_1[\dot{\pi}_1 \circ \sigma]) + m(N_1[\dot{\pi}_1 \circ \tau]) < m(M) + m(N)$ and $m(M_2[\dot{\pi}_2 \circ \sigma]) + m(N_1[\dot{\pi}_2 \circ \tau]) < m(M) + m(N)$ hold. Therefore, we can apply the induction hypothesis to each of these pairs. Furthermore, we have the following chain of equalities.

$$\begin{array}{ll} 1[\dot{\pi}_1 \circ \langle M[\sigma]/A_1 * A_2 \rangle] & \\ = 1[\dot{\pi}_1 \circ \langle M/A_1 * A_2 \rangle \circ \sigma] & \text{substitution equation} \\ = 1[\dot{\pi}_1 \circ \langle M_1 * M_2/A_1 * A_2 \rangle \circ \sigma] & \text{since } M \text{ is } M_1 * M_2 \\ = 1[\dot{\pi}_1 \circ (\langle !, M_1 \rangle * \langle !, M_2 \rangle) \circ \sigma] & \text{by } (*\text{-}\beta) \\ = 1[\langle !, M_1 \rangle \circ \dot{\pi}_1 \circ \sigma] & \\ = M_1[\dot{\pi}_1 \circ \sigma] & \\ = N_1[\dot{\pi}_1 \circ \tau] & \text{by induction hypothesis} \\ = 1[\dot{\pi}_1 \circ \langle N[\sigma]/A_1 * A_2 \rangle] & \text{as for } M \end{array}$$

By similar reasoning, we obtain an analogous equation for M_2 and N_2. We can therefore use (INJECT) to conclude $\Gamma \vdash M[\sigma] = N[\tau] : A = B$, as required. □

The Main Lemma extends to contexts in the expected way.

Lemma 6.5.27. *If* $\vdash \Gamma \Downarrow \Phi$ Bunch *and* $\vdash \Delta \Downarrow \Phi$ Bunch *then* $\vdash \Gamma = \Delta$ Bunch.

Proof. By induction on the derivation of $\vdash \Gamma \Downarrow \Phi$ Bunch.

- Γ is $\diamond$. Then Δ too must be $\diamond$ and the assertion follows by reflexivity.
- Γ is $\Gamma', x\colon A$. Then Φ is $\Phi', x\colon B$, where $\vdash \Gamma' \Downarrow \Phi'$ Bunch and $\Gamma' \vdash A \Downarrow B$ Type. Then Δ must also have the form $\Delta', x\colon C$, where $\vdash \Delta' \Downarrow \Phi'$ Bunch and $\Delta' \vdash C \Downarrow B$ Type. By induction hypothesis we have $\vdash \Gamma' = \Delta'$ Bunch. By (BU-CONV), this implies $\Gamma' \vdash C \Downarrow B$ Type. By Lemma 6.5.26, we furthermore have $\Gamma' \vdash A = C$ Type. By (BU-AEQ) we get $\vdash (\Gamma', x\colon A) = (\Delta', x\colon C)$ Bunch.
- Γ is $\Gamma' * \Gamma''$. Then Φ is $\Phi' * \Phi''$, where $\vdash \Gamma' \Downarrow \Phi'$ Bunch and $\vdash \Gamma'' \Downarrow \Phi''$ Bunch. Then Δ must have the form $\Delta' * \Delta''$ for $\vdash \Delta' \Downarrow \Phi'$ Bunch and $\vdash \Delta'' \Downarrow \Phi''$ Bunch. By induction hypothesis we get $\vdash \Gamma' = \Delta'$ Bunch and $\vdash \Gamma'' = \Delta''$ Bunch. By Lemma 6.4.7 we can apply (BU-MEQ) to get the required $\vdash \Gamma' * \Gamma'' = \Delta' * \Delta''$ Bunch. □

Having established the Main Lemma, we can now define the interpretation of **BT** in **ES**. Thanks to the Main Lemma, this is essentially straightforward.

For the definition of the interpretation of **BT** in **ES**, we need to account for the contexts with a single hole $\Gamma_\circ$ in the system $\mathbf{BT}(*, \Pi, \Pi^*)$. We now introduce notation for that purpose.

Lemma 6.5.28. *Let* $\Gamma'_\circ$ *be a context with a single hole. For any* $\vdash \Phi \Downarrow \Gamma'(\Delta')$ Bunch *there exists a unique* Δ *such that* Φ *is* $\Gamma(\Delta)$ *and* $\vdash \Delta \Downarrow \Delta'$ Bunch.

Proof. By induction on the size of $\Gamma'_\circ$. □

Definition 6.5.29. Given judgements $\vdash \Delta \Downarrow \Delta'$ Bunch, $\vdash \Gamma(\Delta) \Downarrow \Gamma'(\Delta')$ Bunch, $\vdash \Phi \Downarrow \Phi'$ Bunch and $\sigma \Downarrow \chi\colon \Phi \to \Delta$ satisfying $\Gamma_\circ \cap (\Delta \cup \Phi) = \emptyset$, we define a substitution $\Gamma(\sigma)$ and a context-with-hole $\Gamma[\sigma]_\circ$ by mutual induction on the structure of $\Gamma_\circ$ as follows.

$$\Gamma(\sigma) \quad = \quad \begin{cases} \sigma & \text{if } \Gamma_\circ \text{ is } \bigcirc \\ \Gamma'(\sigma) * id & \text{if } \Gamma_\circ \text{ is } \Gamma'_\circ * \Gamma'' \\ id * \Gamma''(\sigma) & \text{if } \Gamma_\circ \text{ is } \Gamma' * \Gamma''_\circ \\ (\Gamma'(\sigma))_\bullet & \text{if } \Gamma_\circ \text{ is } \Gamma'_\circ, x\colon A \end{cases}$$

$$\Gamma[\sigma]_\circ \quad = \quad \begin{cases} \bigcirc & \text{if } \Gamma_\circ \text{ is } \bigcirc \\ \Gamma'[\sigma]_\circ * \Gamma'' & \text{if } \Gamma_\circ \text{ is } \Gamma'_\circ * \Gamma'' \\ \Gamma' * \Gamma''[\sigma]_\circ & \text{if } \Gamma_\circ \text{ is } \Gamma' * \Gamma''_\circ \\ \Gamma'[\sigma]_\circ, x\colon A[\Gamma'(\sigma)] & \text{if } \Gamma_\circ \text{ is } \Gamma'_\circ, x\colon A \end{cases}$$

The next two lemmas are proved by straightforward induction.

Lemma 6.5.30. *Given judgements* $\vdash \Delta \Downarrow \Delta'$ Bunch, $\vdash \Gamma(\Delta) \Downarrow \Gamma'(\Delta')$ Bunch, $\vdash \Phi \Downarrow \Phi'$ Bunch *and* $\sigma \Downarrow \chi \colon \Phi \to \Delta$ *satisfying* $\Gamma_\circ \cap (\Delta \cup \Phi) = \emptyset$*, we can derive* $\vdash \Gamma[\sigma](\Phi) \Downarrow \Gamma'(\Phi')\{\chi\}$ Bunch *and* $\Gamma(\sigma) \Downarrow \chi \colon \Gamma[\sigma](\Phi) \to \Gamma(\Delta)$.

Lemma 6.5.31. *Given judgements* $\vdash \Delta \Downarrow \Delta'$ Bunch, $\vdash \Gamma(\Delta) \Downarrow \Gamma'(\Delta')$ Bunch, $\vdash \Phi \Downarrow \Phi'$ Bunch, $\sigma \colon \Psi \to \Phi$ *and* $\tau \colon \Phi \to \Delta$ *satisfying* $\Gamma_\circ \cap (\Psi \cup \Phi \cup \Delta) = \emptyset$*, we can derive* $\vdash \Gamma[\tau \circ \sigma](\Psi) = \Gamma[\tau][\sigma](\Psi)$ Bunch *and* $\Gamma(\tau \circ \sigma) = \Gamma(\tau) \circ \Gamma[\tau](\sigma) \colon \Gamma[\tau \circ \sigma](\Psi) \to \Gamma(\Delta)$.

We are now ready to give the interpretation of **BT** in **ES**. Of course, we need to assume that **ES** contains enough constants and axioms to interpret the constants and axioms in **BT**. This requirement is captured by the following definition, which states that for each constant in **BT** there exists a corresponding constant of the same name in **ES**.

Definition 6.5.32. Let $\mathscr{T}_{\mathbf{BT}}$ and $\mathscr{T}_{\mathbf{ES}}$ be dependently typed algebraic theories in **BT** and in **ES** respectively. We say that $\mathscr{T}_{\mathbf{BT}}$ and $\mathscr{T}_{\mathbf{ES}}$ are *compatible* if the following conditions hold.

1. If $T \in \mathscr{T}_{\mathbf{BT}}(\Gamma)$ and $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch is derivable, then there exists a derivable judgement $\vdash_{\mathbf{ES}} \Gamma'' \Downarrow |\Gamma|$ Bunch with $T \in \mathscr{T}_{\mathbf{ES}}(\Gamma'')$.

2. If $C \in \mathscr{T}_{\mathbf{BT}}(\Gamma;A)$ and $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch $\Gamma' \vdash_{\mathbf{ES}} A' \Downarrow |A|$ Type are derivable, then there exist derivable judgements $\vdash_{\mathbf{ES}} \Gamma'' \Downarrow |\Gamma|$ Bunch and $\Gamma'' \vdash_{\mathbf{ES}} A'' \Downarrow |A|$ Type with $C \in \mathscr{T}_{\mathbf{ES}}(\Gamma'';A'')$.

3. If $\Gamma \vdash A = B$ Type is in $\mathscr{T}^{\mathscr{E}}_{\mathbf{BT}}(\Gamma)$ and $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch is derivable, then there exist derivable judgements $\vdash_{\mathbf{ES}} \Gamma'' \Downarrow |\Gamma|$ Bunch and $\Gamma'' \vdash_{\mathbf{ES}} A'' \Downarrow |A|$ Type and $\Gamma'' \vdash_{\mathbf{ES}} B'' \Downarrow |B|$ Type such that $\Gamma'' \vdash A'' = B''$ Type is in $\mathscr{T}^{\mathscr{E}}_{\mathbf{ES}}(\Gamma'')$.

4. If $\Gamma \vdash M = N : A$ is in $\mathscr{T}^{\mathscr{E}}_{\mathbf{BT}}(\Gamma;A)$ and $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch and $\Gamma' \vdash_{\mathbf{ES}} A' \Downarrow |A|$ Type are derivable, then there exist derivable judgements $\vdash_{\mathbf{ES}} \Gamma'' \Downarrow |\Gamma|$ Bunch and $\Gamma'' \vdash_{\mathbf{ES}} A'' \Downarrow |A|$ Type and $\Gamma'' \vdash_{\mathbf{ES}} M'' \Downarrow |M| : A''$ and $\Gamma'' \vdash_{\mathbf{ES}} N'' \Downarrow |N| : A''$ such that $\Gamma'' \vdash M'' = N'' : A''$ is in $\mathscr{T}^{\mathscr{E}}_{\mathbf{ES}}(\Gamma'')$.

That this definition is slightly more complicated than perhaps expected is due to the fact that the syntax translation from **ES** to the let-free syntax is integrated with the typing judgement of **ES**. Since in the formulation of a dependently typed algebraic theory we have made no derivability assumptions on the contexts and types, a theory may contain constants for invalid contexts and types. In order to make it possible to satisfy the existential statements in the above definition, the definition is such that one only has to show existence for sufficiently well-formed contexts and types.

The next proposition gives the interpretation of **BT** in **ES**. To each judgement in **BT** we assign a judgement in **ES** such that both judgements have the same translation in the let-free syntax. Lemmas 6.5.26 and 6.5.27 show that if we assign two different **ES**-judgements to the same **BT**-judgement in this way, then the two **ES**-judgements are provably equal. We may therefore speak of *the* interpretation of each **BT**-judgement.

Proposition 6.5.33 (Interpretation)**.** *Let $\mathscr{T}_{\mathbf{BT}}$ and $\mathscr{T}_{\mathbf{ES}}$ be compatible dependently typed algebraic theories in* **BT** *and in* **ES** *respectively and assume that the derivations in* **BT** *and* **ES** *are formed with respect to these theories. Then the following hold.*

1. *If* $\vdash_{\mathbf{BT}} \Gamma$ Bunch *then there exists a derivation of* $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch.
2. *If* $\Gamma \vdash_{\mathbf{BT}} A$ Type *then there exist derivations of* $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch *and* $\Gamma' \vdash_{\mathbf{ES}} A' \Downarrow |A|$ Type.
3. *If* $\Gamma \vdash_{\mathbf{BT}} M : A$ *then there exist derivations of* $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch, $\Gamma' \vdash_{\mathbf{ES}} A' \Downarrow |A|$ Type *and* $\Gamma' \vdash_{\mathbf{ES}} M' \Downarrow |M| : A'$.
4. *If* $\vdash_{\mathbf{BT}} \Gamma = \Delta$ Bunch *then there exist derivations of* $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch, $\vdash_{\mathbf{ES}} \Delta' \Downarrow |\Delta|$ Bunch *and* $\vdash_{\mathbf{ES}} \Gamma' = \Delta'$ Bunch.
5. *If* $\Gamma \vdash_{\mathbf{BT}} A = B$ Type *then there exist derivations of* $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch, $\Gamma' \vdash_{\mathbf{ES}} A' \Downarrow |A|$ Type, $\Gamma' \vdash_{\mathbf{ES}} B' \Downarrow |B|$ Type *and* $\Gamma' \vdash_{\mathbf{ES}} A' = B'$ Type.
6. *If* $\Gamma \vdash_{\mathbf{BT}} M = N : A$ *then there exist derivations of* $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch, $\Gamma' \vdash_{\mathbf{ES}} A' \Downarrow |A|$ Type, $\Gamma' \vdash_{\mathbf{ES}} M' \Downarrow |M| : A'$, $\Gamma' \vdash_{\mathbf{ES}} N' \Downarrow |N| : A'$ *and* $\Gamma' \vdash_{\mathbf{ES}} M' = N' : A'$.

Proof. The proof goes by simultaneous induction on the derivations in **BT**. We use unannotated sequents for judgements in **ES**. For brevity, we write syntax translations at any place in a judgement as a short-hand for derivations of the parts with the corresponding translation. For example, we write $\Gamma' \Downarrow \Gamma \vdash (D \Downarrow |A|) = (E \Downarrow |B|)$ Type as an abbreviation for the four judgements $\vdash \Gamma' \Downarrow |\Gamma|$ Bunch, $\Gamma' \vdash D \Downarrow |A|$ Type, $\Gamma' \vdash E \Downarrow |B|$ Type and $\Gamma' \vdash D = E$ Type.

We proceed by case distinction on the last rule in the **BT**-derivation, showing representative cases.

1. Terms.

   - Type conversion. Suppose the last rule in **BT** is

     $$(\textsc{Ty-Conv})\ \frac{\Gamma \vdash_{\mathbf{BT}} M : A \qquad \Gamma \vdash_{\mathbf{BT}} A = B \text{ Type}}{\Gamma \vdash_{\mathbf{BT}} M : B}$$

     By induction hypothesis we have

     $$\Gamma' \Downarrow |\Gamma| \vdash M' \Downarrow |M| : C \Downarrow |A|, \qquad \Gamma'' \Downarrow |\Gamma| \vdash (D \Downarrow |A|) = (E \Downarrow |B|) \text{ Type}.$$

     By Lemma 6.5.27 we have $\vdash \Gamma' = \Gamma''$ Bunch. Using (Bu-Conv) we get $\Gamma' \vdash D \Downarrow |A|$ Type, from which $\Gamma' \vdash C = D$ Type follows by Lemma 6.5.26. Since using bunch conversion we also have $\Gamma' \vdash D = E$ Type, transitivity implies $\Gamma' \vdash C = E$ Type. Using (Ty-Conv) we obtain the required $\Gamma' \Downarrow |\Gamma| \vdash M' \Downarrow |M| : E \Downarrow |B|$.

   - Substitution. Suppose the last rule in **BT** is

     $$(\textsc{Subst})\ \frac{\Gamma(\Delta, x\colon A) \vdash_{\mathbf{BT}} N : B \qquad \Delta \vdash_{\mathbf{BT}} M : A}{\Gamma(\Delta)\,[M/x] \vdash_{\mathbf{BT}} N\,[M/x] : B\,[M/x]}$$

with $(\Delta \cup \{x\}) \cap \Gamma_\circ = \emptyset$. By induction hypothesis we have $\Phi \Downarrow |\Gamma(\Delta, x\colon A)| \vdash N' \Downarrow |N| : B' \Downarrow |B|$ and $\Delta' \Downarrow |\Delta| \vdash M' \Downarrow |M| : A' \Downarrow |A|$. The context Φ must have the form $\Gamma'(\Delta'', x\colon A'')$ where $\vdash \Delta'' \Downarrow |\Delta|$ Bunch and $\Delta'' \vdash A'' \Downarrow |A|$ Type. Using Lemmas 6.5.27 and 6.5.26, we get $\vdash \Delta' = \Delta''$ Bunch and $\Delta' \vdash A' = A''$ Type. We have the substitution $\langle M' \rangle \Downarrow \{|M|/x\}\colon \Delta' \to \Delta', x\colon A'$. By Lemma 6.5.30, this map can be lifted to

$$\Gamma'(\langle M' \rangle) \Downarrow \{|M|/x\}\colon \Gamma'[\langle M' \rangle](\Delta') \to \Gamma'(\Delta', x\colon A')$$

Since the codomain of this map equals Φ, we can substitute N' along it to get

$$\Gamma'[\langle M' \rangle](\Delta') \vdash N'[\Gamma'(\langle M' \rangle)] \Downarrow |N|\ \{|M|/x\} : B'[\Gamma'(\langle M' \rangle)] \Downarrow |B|\ \{|M|/x\}$$

Since, by Lemma 6.5.30, we have $\vdash \Gamma'[\langle M' \rangle](\Delta') \Downarrow |\Gamma(\Delta)|\ \{|M|/x\}$ Bunch and because $|N|\ \{|M|/x\} = |N[M/x]|$ and $|B|\ \{|M|/x\} = |B[M/x]|$ and $|\Gamma(\Delta)|\ \{|M|/x\} = |\Gamma(\Delta)[M/x]|$ hold, this sequent is just as required.

- Projection. Suppose the last rule in **BT** is

$$(\textsc{Proj})\ \frac{\Gamma \vdash_{\mathbf{BT}} A \text{ Type}}{\Gamma, x\colon A \vdash_{\mathbf{BT}} x : A}$$

  By induction hypothesis we have $\Gamma' \Downarrow |\Gamma| \vdash A' \Downarrow |A|$ Type. Applying the projection rule gives the required $\Gamma', x\colon A' \Downarrow |\Gamma, x\colon A| \vdash 1 \Downarrow |x| : A'[\mathrm{w}] \Downarrow |A|$.

- Constants. The definition of compatibility of dependently typed algebraic theories is such that the induction hypothesis implies the existence of appropriate constants in **ES**.

- Π-Abstraction. Suppose the last rule in **BT** is

$$(\Pi\text{-I})\ \frac{\Gamma, x\colon A \vdash_{\mathbf{BT}} M : B}{\Gamma \vdash_{\mathbf{BT}} \lambda x\colon A.M : \Pi x\colon A.B}$$

  By induction hypothesis we have $\Gamma'' \Downarrow |\Gamma, x\colon A| \vdash M' \Downarrow |M| : C \Downarrow |B|$. By inversion, the context Γ'' must have the form $\Gamma', x\colon A'$ where $\vdash \Gamma' \Downarrow |\Gamma|$ Bunch and $\Gamma' \vdash A' \Downarrow |A|$ Type. We use (Π-I) to obtain $\Gamma' \vdash \lambda A'.M' \Downarrow \lambda x\colon |A|.|M| : \Pi A'.C \Downarrow \Pi x\colon |A|.|B|$. Since we have $\lambda x\colon |A|.|M| = |\lambda x\colon A.M|$ and $\Pi x\colon |A|.|B| = |\Pi x\colon A.B|$, this is as required.

- Π-Application. Suppose the last rule in **BT** is

$$(\Pi\text{-E})\ \frac{\Gamma, x\colon A \vdash_{\mathbf{BT}} B \text{ Type} \qquad \Gamma \vdash_{\mathbf{BT}} M : \Pi x\colon A.B \qquad \Gamma \vdash_{\mathbf{BT}} N : A}{\Gamma \vdash_{\mathbf{BT}} \mathrm{app}_{(x:A)B}(M,N) : B[N/x]}$$

  By induction hypothesis we have

$$\begin{aligned} &\Phi \Downarrow |\Gamma, x\colon A| \vdash D \Downarrow |B| \text{ Type},\\ &\quad \Gamma' \Downarrow |\Gamma| \vdash M' \Downarrow |M| : E \Downarrow |\Pi x\colon A.B|,\\ &\quad \Gamma'' \Downarrow |\Gamma| \vdash N' \Downarrow |N| : C \Downarrow |A|. \end{aligned}$$

  Lemma 6.5.27 gives $\vdash \Gamma' = \Gamma''$ Bunch. By the definition of the syntax translation and inversion, Φ must have the form $\Gamma''', x\colon C'$ with $\Gamma''' \Downarrow |\Gamma| \vdash C' \Downarrow |A|$ Type. Lemmas 6.5.27

and 6.5.26 imply $\vdash \Gamma' = \Gamma'''$ Bunch and $\Gamma' \vdash C = C' \Downarrow |A|$ Type. Using bunch conversion, we can derive $\Gamma' \vdash C \Downarrow |A|$ Type and $\Gamma', x\colon C \vdash D \Downarrow |B|$ Type. Using (Π-Ty), we obtain a derivation of the sequent $\Gamma' \vdash \Pi C.D \Downarrow |\Pi x\colon A.B|$ Type. By Lemma 6.5.26, this implies $\Gamma' \vdash E = \Pi C.D$ Type. Using conversion, we can therefore derive

$$\Gamma' \vdash M' \Downarrow |M| : \Pi C.D \Downarrow |\Pi x\colon A.B|, \qquad \Gamma' \vdash N' \Downarrow |N| : C \Downarrow |A|.$$

Using the rule (Π-E), we obtain $\Gamma' \vdash \mathrm{app}_{[C]D}(M',N') \Downarrow |\mathrm{app}_{(x:A)B}(M,N)| : D[\langle N'\rangle]$. Since we can also derive $\Gamma' \vdash D[\langle N'\rangle] \Downarrow |B[N/x]|$ Type, this completes the case.

- Π^*-Abstraction. Suppose the last rule in **BT** is

$$(\Pi^*\text{-I})\ \frac{\Gamma * x\colon A \vdash_{\mathbf{BT}} M : B}{\Gamma \vdash_{\mathbf{BT}} \lambda^* x\colon A.M : \Pi^* x\colon A.B}$$

By induction hypothesis we have $\Gamma'' \Downarrow |\Gamma * x\colon A| \vdash M' \Downarrow |M| : C \Downarrow |B|$. By inversion, the context Γ'' must have the form $\Gamma' * x\colon A'$ where $\vdash \Gamma' \Downarrow |\Gamma|$ Bunch and $\vdash A' \Downarrow |A|$ Type. We use (Π^*-I) to obtain $\Gamma' \vdash \lambda^* A'.M' \Downarrow \lambda^* x\colon |A|.|M| : \Pi^* A'.C \Downarrow \Pi^* x\colon |A|.|B|$. Since we have $\lambda^* x\colon |A|.|M| = |\lambda^* x\colon A.M|$ and $\Pi^* x\colon |A|.|B| = |\Pi^* x\colon A.B|$, this is as required.

- Π^*-Application. Suppose the last rule in **BT** is

$$(\Pi^*\text{-E})\ \frac{\Gamma * x\colon A \vdash_{\mathbf{BT}} B \text{ Type} \qquad \Gamma \vdash_{\mathbf{BT}} M : \Pi^* x\colon A.B \qquad \Delta \vdash_{\mathbf{BT}} N : A}{\Gamma * \Delta \vdash_{\mathbf{BT}} \mathrm{app}^*_{(x:A)B}(M,N) : B[N/x]}$$

By induction hypothesis we have

$$\begin{aligned} \Phi \Downarrow |\Gamma * x\colon A| \vdash D \Downarrow |B| \text{ Type},\\ \Gamma' \Downarrow |\Gamma| \vdash M' \Downarrow |M| : E \Downarrow |\Pi^* x\colon A.B|,\\ \Delta' \Downarrow |\Delta| \vdash N' \Downarrow |N| : C \Downarrow |A|. \end{aligned}$$

By the definition of the syntax translation and inversion, Φ must have the form $\Gamma''' * x\colon C'$ with $\vdash \Gamma''' \Downarrow |\Gamma|$ Bunch and $\vdash C' \Downarrow |A|$ Type. By Lemma 6.5.27 we obtain $\vdash \Gamma''' = \Gamma'$ Bunch, so that by conversion we have $\Gamma' * x\colon C' \vdash D \Downarrow |B|$ Type. By rule (Π^*-Ty), we obtain $\Gamma' \vdash \Pi^* C'.D \Downarrow |\Pi^* x\colon A.B|$ Type. Lemma 6.5.26 gives $\Gamma' \vdash E = \Pi^* C'.D$ Type. Since using (Subst) we get $\Delta' \vdash C'[!_{\Delta'}] \Downarrow |A|$ Type, Lemma 6.5.26 implies $\Delta' \vdash C'[!_{\Delta'}] = C$ Type. Using conversion, we can derive

$$\Gamma' \vdash M' \Downarrow |M| : \Pi^* C'.D \Downarrow |\Pi^* x\colon A.B|, \qquad \Delta' \vdash N' \Downarrow |N| : C'[!_{\Delta'}].$$

Using (Π^*-E) we get $\Gamma' * \Delta' \vdash \mathrm{app}^*_{[C']D}(M',N') \Downarrow |\mathrm{app}^*_{(x:A)B}(M,N)| : D[id * \langle !, N'\rangle]$. Since we have $\Gamma' * \Delta' \vdash D[id * \langle !, N'\rangle] \Downarrow |B[N/x]|$ Type, this completes the case.

- $*$-Introduction. Suppose the last rule in **BT** is

$$(*\text{-I})\ \frac{\vdash_{\mathbf{BT}} A \text{ Type} \qquad \vdash_{\mathbf{BT}} A \text{ Type} \qquad \Gamma \vdash_{\mathbf{BT}} M : A \qquad \Delta \vdash_{\mathbf{BT}} N : B}{\Gamma * \Delta \vdash_{\mathbf{BT}} \mathrm{pair}^*_{A,B}(M,N) : A * B}$$

By induction hypothesis we have

$$\vdash C \Downarrow |A| \text{ Type} \qquad \vdash D \Downarrow |B| \text{ Type}$$
$$\Gamma' \Downarrow |\Gamma| \vdash M' \Downarrow |M| : E \Downarrow |A| \qquad \Delta' \Downarrow |\Delta| \vdash N' \Downarrow |N| : F \Downarrow |B|$$

Using Lemma 6.5.26 we obtain $\Gamma' \vdash E = C[!]$ Type and $\Delta' \vdash F = D[!]$ Type. Using type conversion and ($*$-I), we can therefore derive the required

$$\Gamma' * \Delta' \Downarrow |\Gamma * \Delta| \vdash \text{pair}^*_{C,D}(M',N') \Downarrow |\text{pair}^*_{A,B}(M,N)| : C*D \Downarrow |A*B|.$$

- $*$-Elimination. Suppose the last rule in **BT** is

$$(*\text{-E})\ \frac{\begin{array}{c}\Gamma(z\colon A*B) \vdash_{\mathbf{BT}} C \text{ Type} \\ \Delta \vdash_{\mathbf{BT}} M : A*B \\ \Gamma(x\colon A * y\colon B)[\text{pair}^*_{A,B}(x,y)/z] \vdash_{\mathbf{BT}} N : C[\text{pair}^*_{A,B}(x,y)/z]\end{array}}{\Gamma(\Delta)[M/z] \vdash_{\mathbf{BT}} \text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } N : C[M/z]}$$

We can deduce $\Gamma_\circ \cap \{x,y,z\} = \emptyset$ from the rule. The induction hypothesis provides us with derivations of

$$\Phi \Downarrow |\Gamma(z\colon A*B)| \vdash C' \Downarrow |C| \text{ Type},$$
$$\Delta' \Downarrow |\Delta| \vdash M' \Downarrow |M| : A'*B' \Downarrow |A*B|,$$
$$\Psi \Downarrow |\Gamma(x\colon A * y\colon B)[\text{pair}^*_{A,B}(x,y)/z]| \vdash N' \Downarrow |N| : D \Downarrow |C[\text{pair}^*_{A,B}(x,y)/z]|.$$

First, Φ has the form $\Gamma'(z\colon A''*B'')$ where $\vdash A''*B'' \Downarrow |A*B|$ Type. Because of Lemma 6.5.26 and conversion, we can assume that A'' is A' and B'' is B'. Let $\sigma \Downarrow \chi\colon (x\colon A') * (y\colon B') \to z\colon A'*B'$, in which χ is $\{\text{pair}^*_{|A|,|B|}(x,y)/z\}$, be the evident substitution. Because we have $\Gamma_\circ \cap \{x,y,z\} = \emptyset$, it can be lifted to

$$\Gamma'(\sigma) \Downarrow \chi\colon \Gamma'[\sigma](x\colon A' * y\colon B') \to \Gamma'(z\colon A'*B').$$

We have $\langle M'/A'*B'\rangle \Downarrow \xi : \Delta' \to (x\colon A') * (y\colon B')$, where ξ is $\{r_{|A|,|B|}(|M|)/y\}\{l_{|A|,|B|}(|M|)/x\}$. This map can also be lifted:

$$\Gamma'[\sigma](\langle M'/A'*B'\rangle) \Downarrow \xi : \Gamma'[\sigma][\langle M'/A'*B'\rangle](\Delta') \to \Gamma'[\sigma](x\colon A' * y\colon B')$$

By ($*$-η), we have $\sigma \circ \langle M'/A'*B'\rangle = \langle !_{\Delta'}, M'\rangle$, so that using Lemma 6.5.31 and conversion we get

$$\Gamma'[\sigma](\langle M'/A'*B'\rangle) \Downarrow \xi : \Gamma'[\langle !_{\Delta'}, M'\rangle](\Delta') \to \Gamma'[\sigma](x\colon A' * y\colon B').$$

Now we have $\vdash \Gamma'[\sigma](x\colon A' * y\colon B') \Downarrow |\Gamma(x\colon A * y\colon B)[\text{pair}^*_{A,B}(x,y)/z]|$ Bunch, from which we obtain $\vdash \Gamma'[\sigma](x\colon A' * y\colon B') = \Psi$ Bunch by Lemma 6.5.27. Since Φ is $\Gamma'(z\colon A'*B')$, we can substitute C' along the map $\Gamma'(\sigma)$ to get $\Psi \vdash C'[\Gamma'(\sigma)] \Downarrow |C[\text{pair}^*_{A,B}(x,y)/z]|$ Type. By Lemma 6.5.26, we get $\Psi \vdash D = C'[\Gamma'(\sigma)]$ Type. By substituting in this equation, we get

$$\Gamma'[\langle !_{\Delta'}, M'\rangle](\Delta') \vdash D[\Gamma'[\sigma](\langle M'/A'*B'\rangle)] = C'[\Gamma'(\sigma)][\Gamma'[\sigma](\langle M'/A'*B'\rangle)] \text{ Type}.$$

Using $\Gamma'(\sigma)\circ\Gamma'[\sigma](\langle M'/A'*B'\rangle) = \Gamma'(\sigma\circ\langle M'/A'*B'\rangle)$ and the rule $(*\text{-}\eta)$, we get

$$\Gamma'[\langle !_{\Delta'}, M'\rangle](\Delta') \vdash D[\Gamma'[\sigma](\langle M'/A'*B'\rangle)] = C'[\Gamma'(\langle !_{\Delta'}, M'\rangle)] \text{ Type}.$$

Therefore, by substituting N' along $\Gamma'[\sigma](\langle M'/A'*B'\rangle)$, we can derive

$$\Gamma'[\langle !_{\Delta'}, M'\rangle](\Delta') \vdash N'[\Gamma'[\sigma](\langle M'/A'*B'\rangle)] \Downarrow |N|\ \xi : C'[\Gamma'(\langle !_{\Delta'}, M'\rangle)].$$

Finally, the equations $(|N|\ \xi) = |\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } N|$ and $\Gamma'[\langle !_{\Delta'}, M'\rangle](\Delta') \Downarrow |\Gamma(\Delta)[M/z]|$ and $C'[\Gamma'(\langle !_{\Delta'}, M'\rangle)] \Downarrow |C[M/z]|$ are straightforward to show. Therefore, the sequent that we have just derived is as required.

2. Term equations.

- $(\Pi\text{-}\beta)$. Suppose the last rule in **BT** is

$$(\Pi\text{-}\beta)\ \frac{\Gamma, x\colon A \vdash_{\mathbf{BT}} M : B \qquad \Gamma \vdash_{\mathbf{BT}} N : A}{\Gamma \vdash_{\mathbf{BT}} \mathrm{app}_{(x:A)B}(\lambda x\colon A.M, N) = M[N/x] : B[N/x]}$$

  By induction hypothesis we have $\Gamma' \Downarrow |\Gamma, x\colon A| \vdash M' \Downarrow |M| : B' \Downarrow |B|$ and $\Gamma'' \Downarrow |\Gamma| \vdash N' \Downarrow |N| : A' \Downarrow |A|$. The context Γ' must have the form $\Gamma''', x\colon A''$ where $\vdash \Gamma''' \Downarrow |\Gamma|$ Bunch and $\Gamma''' \vdash A'' \Downarrow |A|$ Type. Using Lemma 6.5.26, we can derive $\vdash \Gamma'' = \Gamma'''$ Bunch and $\Gamma'' \vdash A'' = A'$ Type. Hence, using (Bu-Conv), we obtain $\Gamma'', x\colon A' \Downarrow |\Gamma, x\colon A| \vdash M' \Downarrow |M| : B' \Downarrow |B|$. By substitution and $(\Pi\text{-E})$, we have

$$\Gamma' \vdash M'[\langle N'\rangle] \Downarrow |M[N/x]| : B'[\langle N'\rangle] \Downarrow |B[N/x]|,$$
$$\Gamma' \vdash \mathrm{app}_{[A']B'}(\lambda A'.M', N') \Downarrow |\mathrm{app}_{(x:A)B}(\lambda x\colon A.M, N)| : B'[\langle N'\rangle].$$

  The case is completed by using $(\Pi\text{-}\beta)$ to get $\Gamma' \vdash \mathrm{app}_{[A']B'}(\lambda A'.M', N') = M'[\langle N'\rangle] : B'[\langle N'\rangle]$.

- $(*\text{-}\eta)$. Suppose the last rule in **BT** is

$$(*\text{-}\eta)\ \frac{\Delta \vdash_{\mathbf{BT}} M : A{*}B \qquad \Gamma(z\colon A{*}B) \vdash_{\mathbf{BT}} N : C}{\Gamma(\Delta)[M/z] \vdash_{\mathbf{BT}} N[M/z] = \text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } (N[\mathrm{pair}^*_{A,B}(x,y)/z]) : C[M/z]}$$

  Since all contexts in this rule are assumed to declare each variable at most once, we have $\Delta_\circ \cap \{x,y,z\} = \emptyset$. The induction hypothesis yields:

$$\Delta' \Downarrow |\Delta| \vdash M' \Downarrow |M| : D \Downarrow |A{*}B|,$$
$$\Phi \Downarrow |\Gamma(z\colon A{*}B)| \vdash N' \Downarrow |N| : E \Downarrow |C|.$$

  The context Φ must have the form $\Gamma'(z\colon A'{*}B')$ where $\vdash A' \Downarrow |A|$ Type and $\vdash B' \Downarrow |B|$ Type. Using Lemma 6.5.26, we can show $\Delta' \vdash D = A'{*}B'$ Type. Let

$$\sigma \Downarrow \{|\mathrm{pair}^*_{A,B}(x,y)|/z\}\colon (x\colon A') * (y\colon B') \to z\colon A'{*}B'$$

  be the evident map. Since we have $\Delta_\circ \cap \{x,y,z\}$, this map can be lifted to

$$\Gamma'(\sigma) \Downarrow \{|\mathrm{pair}^*_{A,B}(x,y)|/z\}\colon \Gamma'[\sigma](x\colon A' * y\colon B') \to \Gamma'(z\colon A'{*}B').$$

Furthermore, we have

$$\Gamma'[\sigma](\langle M'/A'*B'\rangle) \Downarrow \chi : \Gamma'[\sigma][\langle M'/A'*B'\rangle](\Delta') \to \Gamma'[\sigma](x\colon A'*y\colon B'),$$

where χ is $\{r_{|A|,|B|}(|M|)/y\}\{l_{|A|,|B|}(|M|)/x\}$. We have the equations $\Gamma'[\sigma][\langle M'/A'*B'\rangle](\Delta') = \Gamma'[\sigma\circ\langle M'/A'*B'\rangle](\Delta')$ and $\Gamma'(\sigma)\circ\Gamma'[\sigma](\langle M'/A'*B'\rangle) = \Gamma'(\sigma\circ\langle M'/A'*B'\rangle)$. Write τ as an abbreviation for $\sigma\circ\langle M'/A'*B'\rangle$. Using (SUBST) we can derive

$$\Gamma'[\tau](\Delta') \vdash E[\Gamma'(\tau)]] \Downarrow |\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } (C[\text{pair}^*_{A,B}(x,y)/z])| \text{ Type}, \tag{6.2}$$

$$\Gamma'[\tau](\Delta') \vdash N'[\Gamma'(\tau)] \Downarrow |\text{let } M \text{ be } x{:}A{*}y{:}B \text{ in } (N[\text{pair}^*_{A,B}(x,y)/z])| : E[\Gamma'(\tau)], \tag{6.3}$$

$$\Gamma'[\langle !, M'\rangle](\Delta') \vdash N'[\Gamma'(\langle !, M'\rangle)] \Downarrow |N[M/z]| : E[\Gamma'(\langle !, M'\rangle)] \Downarrow |C[M/z]|. \tag{6.4}$$

The terms in (6.3) and (6.4) correspond to the two terms in the equation $(*\text{-}\eta)$ above. It therefore suffices to show that they are equal.

We have the following commuting diagram, in which we use the suggestive notation $x*y$ for the evident term. Notice that $\sigma = !_\bullet \circ \langle x*y\rangle$ holds by definition.

$$\begin{array}{ccc}
u\colon A'*B' & \xrightarrow{\langle 1/A'*B'\rangle} & x\colon A'*y\colon B' \\
{\scriptstyle \langle x*y[\langle 1/A'*B'\rangle]}\downarrow & & \downarrow{\scriptstyle \langle x*y\rangle} \\
u\colon A'*B', z\colon A'*B' & \xrightarrow[\langle 1/A'*B'\rangle_\bullet]{} & (x\colon A'*y\colon B'), z\colon A'*B' \\
{\scriptstyle !_\bullet}\downarrow & & \downarrow{\scriptstyle !_\bullet} \\
z\colon A'*B' & = & z\colon A'*B'
\end{array}$$

Therefore, we have $\sigma\circ\langle 1/A'*B'\rangle = !_\bullet \circ \langle (x*y)[\langle 1/A'*B'\rangle]\rangle$, from which using $(*\text{-}\eta)$ we get $\sigma\circ\langle 1/A'*B'\rangle = !_\bullet\circ\langle 1\rangle$. Hence,

$$\begin{aligned}
\tau &= \sigma\circ\langle M'/A'*B'\rangle \\
&= \sigma\circ\langle 1/A'*B'\rangle\circ\langle !, M'\rangle \\
&= !_\bullet\circ\langle 1\rangle\circ\langle !, M'\rangle \\
&= !_\bullet\circ\langle 1\rangle\circ\langle !, M'\rangle \\
&= !_\bullet\circ\langle !, M'\rangle_\bullet\circ\langle 1[\langle !, M'\rangle]\rangle \\
&= !_\bullet\circ\langle M'\rangle \\
&= \langle !, M'\rangle
\end{aligned}$$

Using this equality, we can derive that the two above statements are equal:

$$\Gamma[\tau](\Delta') \vdash N'[\Gamma'(\sigma\circ\langle M'/A'*B'\rangle)] = N'[\Gamma'(\langle !, M'\rangle)] : E[\Gamma'(\langle !, M'\rangle)]$$

If we look at the syntactic translations of these terms and types we see that they correspond to the terms in $(*\text{-}\eta)$ above, so that the derived equation is as required. □

6.6 Discussion and Further Work

In the proof that different derivations of the same sequent have equal interpretations (Lemma 6.5.26), we have made essential use of the assumption that $*$ is a strict affine symmetric monoidal structure. However, it is reasonable to expect that, perhaps subject to some modifications, **BT** is also a sound theory for an (affine) symmetric monoidal structure $\otimes$. The main obstacle in proving soundness in this case is to show that different derivations of $\mathrm{pair}^{\otimes}_{A,B}(M,N)$ and (let M be $x\colon A \otimes y\colon B$ in N) respectively have equal interpretations. This should be straightforward in the case where the type theory has strengthening. Indeed, it appears that strengthening is needed to show soundness for the interpretation of the $\alpha\lambda$-calculus of O'Hearn and Pym, in particular for showing that two derivations of the same judgement have the same interpretation. For example, suppose we can derive $x\colon A * y\colon B \vdash M : C$ and $x\colon A * u\colon D \vdash M : C$ and $u\colon D * v\colon E \vdash N : F$ and $y\colon B * v\colon E \vdash N : F$ but not $x\colon A \vdash M : C$ or $v\colon E \vdash N : F$. Then we can derive both $(x\colon A * y\colon B) * (u\colon D * v\colon E) \vdash M * N : C * F$ and $(x\colon A * u\colon D) * (y\colon B * v\colon E) \vdash M * N : C * F$, and using the structural rules we can transform the second sequent into the first one. There appears to be no reason why the interpretations of these two derivations should be equal. With strengthening, on the other hand, this is straightforward, since both sequents 'come from' $(x\colon A) * (v\colon E) \vdash M * N : C * F$. This, of course, is not a problem for the $\alpha\lambda$-calculus, since strengthening is easy to prove. For dependent types, however, proving strengthening is quite complicated. Moreover, the strengthening property need not even hold for all dependently typed algebraic theories, as we have observed in Chapter 4. We therefore prefer to develop the interpretation for strict affine symmetric monoidal structures only, rather than pre-conditioning the interpretation on the strengthening property.

Since the problems with showing soundness without the assumption that $*$ is strict affine arise only because of $*$-types, we believe that it is straightforward to show soundness of $\mathbf{BT}(1,\Sigma,\Pi,\Pi^*)$ without the assumption of strict affineness.

Regarding the technical development in this chapter, one may be dissatisfied with the presence of the translation $|-|$ to the let-free syntax. We have used it to define a translation from **BT** to **ES**. In the next chapter we show that **ES** can be translated to **BT** by constructing a term model from **BT**. This gives us the situation

$$\mathbf{BT} \Longleftrightarrow \mathbf{ES}.$$

The presence of $|-|$ in the translation from **BT** to **ES** suggests that it may be possible to refine this picture to

$$\mathbf{BT} \Longleftrightarrow \mathbf{BT}^{-} \Longleftrightarrow \mathbf{ES},$$

where $\mathbf{BT}^{-}$ is a type theory with affine projections $A * B \to A$ and $A * B \to B$ instead of let-terms. Not only would such a refinement clarify the use of $|-|$, it could also be helpful in giving an algorithm for deciding equality of $\mathbf{BT}^{-}$. As discussed at the end of Chapter 5, deciding equality in $\mathbf{BT}^{-}$ should be simpler than in **BT**, since there are no commuting conversions in $\mathbf{BT}^{-}$. Showing an equivalence between **BT** and $\mathbf{BT}^{-}$ would therefore help to simplify the treatment of equations in **BT**. We leave it as future work to find out if it is possible to give a type theory $\mathbf{BT}^{-}$ such that the above situation holds.

Chapter 7

Completeness

In this chapter we show completeness of $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ for the structure from Section 6.1 by construction of a term model. We give some direct consequences of completeness, such as unicity of typing and that all commuting conversions are derivable. Suggested by these consequences, we observe further properties of $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$, such as admissibility of substitution up to commuting conversion.

Unless otherwise stated, all the judgements in this chapter are made in system $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$.

7.1 A Term Model

We use the syntax of $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ to construct a $(*,1,\Sigma,\Pi,\Pi^*)$-type category (Definition 6.1.1). The main part of a $(*,1,\Sigma,\Pi,\Pi^*)$-type category is a fibration $p\colon \mathbb{E}\to\mathbb{B}$. We start this section by building such a fibration from the syntax. To do so, we first define categories $\mathbb{B}_0$ and $\mathbb{E}_0$ of syntactic contexts and types-in-context. Then we define categories $\mathbb{B}$ and $\mathbb{E}$ by identifying the objects in $\mathbb{B}_0$ and $\mathbb{E}_0$ up to bijective renaming of variables in the contexts. With these ingredients we define a fibration $p\colon \mathbb{E}\to\mathbb{B}$. The rest of this section is then devoted to showing that p has the structure of a $(*,1,\Sigma,\Pi,\Pi^*)$-type category.

Define a category $\mathbb{B}_0$ as follows.

Objects. The objects of $\mathbb{B}_0$ are derivable bunches $\vdash \Gamma$ Bunch, identified up to provable equality, i.e. if $\vdash \Gamma = \Delta$ Bunch then Γ and Δ are the same object.

Morphisms. The morphisms of $\mathbb{B}_0$ from Γ to Δ are the substitutions $\sigma\colon \Gamma\to\Delta$. Two substitutions σ and τ denote the same morphism if $\sigma=\tau\colon \Gamma\to\Delta$ holds.

The identity morphism is the substitution $\langle\rangle$ and composition is given by composition of substitutions. Lemma 5.5.4 shows that $\mathbb{B}_0$ is a category.

Next define a category $\mathbb{E}_0$ as follows.

Objects. The objects are derivable types $\Gamma \vdash A$ Type, identified up to provable equality, that is if we can derive $\vdash \Gamma = \Delta$ Bunch and $\Gamma \vdash A = B$ Type then $\Gamma \vdash A$ Type and $\Delta \vdash B$ Type are the same object.

Morphisms. The morphisms from $\Gamma \vdash A$ Type to $\Delta \vdash B$ Type are pairs (σ, M) of a substitution $\sigma \colon \Gamma \to \Delta$ and a term $\Gamma, x \colon A \vdash M : B[\sigma]$, where x is a fresh variable. We identify morphisms up to provable equality, i.e. $(\sigma, M) = (\tau, N)$ if $\sigma = \tau \colon \Gamma \to \Delta$ and $\Gamma, x \colon A \vdash M = N : B[\sigma]$, and we identify morphisms if they differ only in the name of the variable x, i.e. $(\sigma, M) = (\sigma, M[y/x])$ for any fresh variable y.

The identity morphism on $\Gamma \vdash A$ Type is given by the pair $(\langle\rangle, x)$ where $\Gamma, x \colon A \vdash x : A$. The composition of maps $(\sigma, M) \colon (\Gamma \vdash A \text{ Type}) \to (\Delta \vdash B \text{ Type})$, where $\Gamma, x \colon A \vdash M : B[\sigma]$, and $(\tau, N) \colon (\Delta \vdash B \text{ Type}) \to (\Phi \vdash C \text{ Type})$, where $\Delta, y \colon B \vdash N : C[\tau]$, is defined to be the pair $(\tau \circ \sigma, N[\sigma][M/y])$, where $\Gamma, x \colon A \vdash N[\sigma][M/y] : C[\tau][\sigma]$.

Lemma 7.1.1. *$\mathbb{E}_0$ is a category.*

Proof. It is straightforward to show that $(\langle\rangle, x)$ is a unit for composition. For associativity, suppose that we have three morphisms $(\sigma, M) \colon (\Gamma \vdash A \text{ Type}) \to (\Delta \vdash B \text{ Type})$, where $\Gamma, x \colon A \vdash M : B[\sigma]$, $(\tau, N) : (\Delta \vdash B \text{ Type}) \to (\Phi \vdash C \text{ Type})$, where $\Delta, y \colon B \vdash N : C[\tau]$, and $(\rho, R) \colon (\Phi \vdash C \text{ Type}) \to (\Psi \vdash D \text{ Type})$, where $\Phi, z \colon C \vdash R : D[\rho]$. We can assume that x, y and z are fresh variables. By definition, we have

$$(\rho, R) \circ ((\tau, N) \circ (\sigma, M)) = (\rho \circ (\tau \circ \sigma), R[\tau \circ \sigma][N[\sigma][M/y]/z]),$$

where $\Gamma, x \colon A \vdash R[\tau \circ \sigma][N[\sigma][M/y]/z] : D[\rho][\tau \circ \sigma]$. On the other hand,

$$((\rho, R) \circ (\tau, N)) \circ (\sigma, M) = ((\rho \circ \tau) \circ \sigma, R[\tau][N/z][\sigma][M/y]),$$

where $\Gamma, x \colon A \vdash R[\tau][N/z][\sigma][M/y] : D[\rho \circ \tau][\sigma]$. It remains to show that the two terms in these two compositions are provably equal. By Lemma 5.5.5 and the definition of equality for substitutions, we have $R[\tau][N/z][\sigma][M/y] = R[\tau][\sigma][N[\sigma]/z][M/y]$. Moreover, since y can be assumed to be fresh for R, τ and σ, we can use the normal syntactic substitution lemma to obtain $R[\tau][\sigma][N[\sigma]/z][M/y] = R[\tau][\sigma][N[\sigma][M/y]/z]$. But this shows that the two terms are equal, thus completing the proof. □

In the definition of the categories $\mathbb{B}_0$ and $\mathbb{E}_0$ we have not identified bunches up to variable renaming. While this simplifies the definitions and makes it easier to show that $\mathbb{B}_0$ and $\mathbb{E}_0$ are categories, it makes the definition of the categorical structure of $\mathbb{B}_0$ and $\mathbb{E}_0$ harder. For example, we would like to define a monoidal structure $*$ on $\mathbb{B}_0$ mapping two objects Γ and Δ to the object $\Gamma * \Delta$. However, the context $\Gamma * \Delta$ is not always defined, since $v(\Gamma)$ and $v(\Delta)$ need not be disjoint. Another problem is that the definition of the comprehension functor taking $(\Gamma \vdash A \text{ Type})$ to $(\Gamma, x \colon A)$ depends on the choice of a fresh variable x. To address these problems, we now take the quotient of $\mathbb{B}_0$ and $\mathbb{E}_0$ with respect to bijective variable renaming. Another possible solution would be to assume a choice function that, for each object Γ, returns a name that is fresh for Γ. Then we could define $*$ such that Γ and Δ are mapped to $\Gamma' * \Delta$, where Γ' is given by replacing the variables in Γ with freshly chosen ones.

We define the categories $\mathbb{B}$ and $\mathbb{E}$ by taking quotients of $\mathbb{B}_0$ and $\mathbb{E}_0$. In the formulation of the equivalence relation with respect to which the quotient is taken, we use bijective variable renamings α and β. We write $\Gamma[\alpha]$ for the action of the renaming α on Γ. Each renaming α can be made into a substitution $\alpha\colon \Gamma[\alpha] \to \Gamma$ in the evident way.

Objects. The objects of $\mathbb{B}$ are equivalence classes of objects of $\mathbb{B}_0$ under the equivalence relation $\sim$ generated by $\Gamma \sim \Gamma[\alpha]$, where α is a bijective renaming of variables.

Morphisms. The morphisms of $\mathbb{B}$ from $[\Gamma]_\sim$ to $[\Delta]_\sim$ are equivalence classes of morphisms of $\mathbb{B}_0$ under the equivalence relation $\sim$ generated by $(\sigma\colon \Gamma \to \Delta) \sim (\alpha \circ \sigma \circ \beta\colon \Gamma[\beta] \to \Delta[\alpha^{-1}])$, where α and β are bijective renamings of variables.

The category $\mathbb{E}$ is defined similarly to $\mathbb{B}$:

Objects. The objects of $\mathbb{E}$ are equivalence classes of objects of $\mathbb{E}_0$ under the equivalence relation $\sim$ generated by $(\Gamma \vdash A \text{ Type}) \sim (\Gamma[\alpha] \vdash A[\alpha] \text{ Type})$, where α is a bijective renaming of variables.

Morphisms. The morphisms of $\mathbb{E}$ from $[\Gamma \vdash A \text{ Type}]_\sim$ to $[\Delta \vdash B \text{ Type}]_\sim$ are equivalence classes of morphisms of $\mathbb{E}_0$ under the equivalence relation $\sim$ generated by $((\sigma, M)\colon (\Gamma \vdash A \text{ Type}) \to (\Delta \vdash B \text{ Type})) \sim ((\alpha \circ \sigma \circ \beta, M[\beta])\colon (\Gamma[\beta] \vdash A[\beta] \text{ Type}) \to (\Delta[\alpha^{-1}] \vdash B[\alpha^{-1}] \text{ Type}))$.

In short, the above definitions amount to the usual identification of contexts, types and terms up to renaming of variables in contexts, as in e.g. [56, 48]. As in loc. cit. we make the equivalence classes in $\mathbb{B}$ and $\mathbb{E}$ implicit, i.e. we denote the equivalence classes by representatives only. Furthermore, it is straightforward to show that we can always assume variable names in the representative to be fresh.

Lemma 7.1.2. *$\mathbb{B}$ is a category and is equivalent to $\mathbb{B}_0$.*

Proof. By construction, $\mathbb{B}$ is just the quotient category $\mathbb{B}_0/\mathbb{R}$, where $\mathbb{R}$ is the category having the same objects as $\mathbb{B}_0$ but as morphisms only bijective renamings. It is straightforward to show that $\mathbb{R}$ contains at most one morphism between any two objects and that all its morphisms are bijections. Using this, it is a standard result, see e.g. [70], that $\mathbb{B}_0/\mathbb{R}$ is a category and that the evident functor $\mathbb{B}_0 \to \mathbb{B}_0/\mathbb{R}$ is an equivalence. □

A similar argument applies to $\mathbb{E}$.

Lemma 7.1.3. *$\mathbb{E}$ is a category and is equivalent to $\mathbb{E}_0$.*

Lemma 7.1.4. *The category $\mathbb{B}$ has an affine monoidal structure given by the functor mapping two objects Γ and Δ with $FV(\Gamma) \cap FV(\Delta) = \emptyset$ (which can always be assumed by the identification up to bijective renamings) to $\Gamma * \Delta$ and mapping two morphisms $\sigma\colon \Gamma \to \Delta$ and $\tau\colon \Phi \to \Psi$ with $FV(\Gamma, \Delta, \sigma) \cap FV(\Phi, \Psi, \tau) = \emptyset$ (which can always be assumed by the identification up to bijective renamings) to the map $\sigma \circ \tau\colon \Gamma * \Phi \to \Delta * \Psi$.*

Proof. Because of the rule (SUB-STR), we can define the coherent isomorphisms as $\langle\rangle\colon (\Gamma * \diamond) \to \Gamma$, $\langle\rangle\colon (\Gamma * \Delta) \to (\Delta * \Gamma)$ and $\langle\rangle\colon (\Gamma * \Delta) * \Phi \to \Gamma * (\Delta * \Phi)$. Naturality of these isomorphisms, the coherence equations and functoriality of $*$ follow using Lemma 5.5.7. □

Next we define a functor $p\colon \mathbb{E} \to \mathbb{B}$. It maps an object $(\Gamma \vdash A \text{ Type})$ in $\mathbb{E}$ to the object Γ in $\mathbb{B}$ and a morphism (σ, M) in $\mathbb{E}$ to the morphism σ in $\mathbb{B}$.

Lemma 7.1.5. *The functor p defines a fibration.*

Proof. We have to show that, for each object $(\Delta \vdash B \text{ Type})$ in $\mathbb{E}$ and each map $\sigma\colon \Gamma \to \Delta$ in $\mathbb{B}$ there exists in $\mathbb{E}$ a cartesian morphism $(\Gamma \vdash A \text{ Type}) \to (\Delta \vdash B \text{ Type})$ over σ.

We show that the morphism $(\sigma, x)\colon (\Gamma \vdash B[\sigma] \text{ Type}) \to (\Delta \vdash B \text{ Type})$, where $\Gamma, x\colon B[\sigma] \vdash x : B[\sigma]$, is cartesian over σ. To this end assume maps $\tau\colon \Phi \to \Delta$ and $\rho\colon \Phi \to \Gamma$ satisfying $\sigma \circ \rho = \tau$, and a map $(\tau, M)\colon (\Phi \vdash C \text{ Type}) \to (\Delta \vdash B \text{ Type})$ over τ, where $\Phi, y\colon C \vdash M : B[\tau]$. We have to show that there exists a unique map (ρ, N) over ρ, where $\Phi, z\colon C \vdash N : B[\sigma][\rho]$, satisfying $(\sigma, x) \circ (\rho, N) = (\tau, M)$. If such a map exists then, by definition of composition, we must have $\Phi, z\colon C \vdash x[\rho][N/x] = M : B[\sigma][\rho] = B[\tau]$. Since x can be assumed to be fresh, we have $\Phi, x\colon B[\sigma][\rho] \vdash x[\rho] = x : B[\sigma][\rho]$ by Lemma 5.5.6. Hence, $\Phi, z\colon C \vdash N = x[N/x] = x[\rho][N/x] = M : B[\sigma][\rho]$ holds. This shows uniqueness of (ρ, N). On the other hand, using $\Phi, z\colon C \vdash M : B[\tau] = B[\sigma][\rho]$, we also have that (ρ, M) is a morphism from $\Phi \vdash C$ Type to $\Gamma \vdash B[\sigma]$ Type, which shows existence. □

Lemma 7.1.6. *The fibration p becomes a split fibration if, for $\sigma\colon \Gamma \to \Delta$ in $\mathbb{B}$ and $(\Delta \vdash B \text{ Type})$ in $\mathbb{E}_\Delta$, we choose the cartesian lifting $\overline{\sigma}(\Delta \vdash B \text{ Type})$ to be the morphism (σ, x) from $(\Gamma \vdash B[\sigma] \text{ Type})$ to $(\Delta \vdash B \text{ Type})$.*

Proof. It is immediate that the lifting of the identity is the identity. The lifting of a composition $\sigma \circ \tau$ is defined to be $(\sigma \circ \tau, x)$. The composition of the lifted maps $(\sigma, x) \circ (\tau, x)$, on the other hand, is defined to be $(\sigma \circ \tau, x[\sigma])$. That this equals $(\sigma \circ \tau, x)$ follows by Lemma 5.5.6. Note that we can assume x to be sufficiently fresh. □

Lemma 7.1.7. *The split fibration p has split fibred terminal objects.*

Proof. We have to show that the split fibred functor p from the identity fibration $id\colon \mathbb{B} \to \mathbb{B}$ to $p\colon \mathbb{E} \to \mathbb{B}$ has a fibred right adjoint $1\colon \mathbb{B} \to \mathbb{E}$. Define $1(\Gamma) = (\Gamma \vdash 1 \text{ Type})$ and $1(\sigma\colon \Gamma \to \Delta) = \overline{\sigma}(1(\Delta))$. It is evident that this defines a split fibred functor from id to p, i.e. that $id = p \circ 1$ holds and that 1 preserves the splitting. The adjunction $p \dashv 1$ with vertical unit and counit follows easily from the fact that $\Gamma \vdash M : 1$ implies $\Gamma \vdash M = \text{unit} : 1$. □

Lemma 7.1.8. *The split fibration p has comprehension, i.e. the terminal object functor $1\colon \mathbb{B} \to \mathbb{E}$ has a right adjoint $\{-\}\colon \mathbb{E} \to \mathbb{B}$.*

Proof. Define the functor $\{-\}$ to map an object $(\Gamma \vdash A \text{ Type})$ in $\mathbb{E}$ to the object $(\Gamma, x\colon A)$ in $\mathbb{B}$ and a morphism $(\sigma, M)\colon (\Gamma \vdash A \text{ Type}) \to (\Delta \vdash B \text{ Type})$ in $\mathbb{E}$, where $\Gamma, y\colon A \vdash M : B[\sigma]$, to the morphism $\sigma \circ \langle M/x\rangle\colon (\Gamma, y\colon A) \to (\Delta, x\colon B)$ in $\mathbb{B}$, where in both cases x and y are some/any fresh variables. This is well-defined because the objects and morphisms are identified under bijective renaming. Note also that we can derive $\sigma\colon (\Gamma, y\colon A,\ x\colon B[\sigma]) \to (\Delta, x\colon B)$ using (SUB-LIFT) and (SUB-STR).

To show $1 \dashv \{-\}$, we define the unit of the adjunction $\eta_\Gamma \stackrel{\text{def}}{=} \langle \text{unit}/y\rangle\colon \Gamma \to (\Gamma, y\colon 1)$ for some/any fresh y. It is routine to show that η is natural. For universality of η we have to show that, for all $\sigma\colon \Gamma \to (\Delta, x\colon A)$ in $\mathbb{B}$, there exists a unique $\sigma^\sharp\colon (\Gamma \vdash 1 \text{ Type}) \to (\Delta \vdash A \text{ Type})$ in $\mathbb{E}$, given by, say, $\tau\colon \Gamma \to \Delta$ and $\Gamma, y\colon 1 \vdash M : A[\tau]$, such that $\sigma = \tau \circ \langle M/x\rangle \circ \eta_\Gamma\colon \Gamma \to (\Delta, x\colon A)$ holds for some/any fresh x and y. Suppose we have such a morphism $\sigma^\sharp$ given by τ and M. First, we show $\sigma = \tau\colon \Gamma \to \Delta$. To this end, we have to show that, for all contexts-with-hole $\Phi_\circ$ that declare only fresh names and all terms $\Phi(\Delta) \vdash R : C$, we have $(\Phi_\circ[\sigma])(\Gamma) \vdash R[\sigma] = R[\tau] : C[\sigma] = C[\tau]$. Since $\tau\colon \Gamma \to \Delta$, it is straightforward to show that neither $C[\tau]$ nor $R[\tau]$ contain x or y as a free variable. We can use the equality $\sigma = \tau \circ \langle M/x\rangle \circ \eta_\Gamma\colon \Gamma \to (\Delta, x\colon A)$ for the term $\Phi(\Delta, x\colon A) \vdash R : C$ to give us $(\Phi_\circ[\sigma])(\Gamma) \vdash R[\sigma] = R[\tau][M/x][\text{unit}/y] : C[\sigma] = C[\tau][M/x][\text{unit}/y]$. Since x and y are not free in $R[\tau]$ and $C[\tau]$, this is just as required to give us $(\Phi_\circ[\sigma])(\Gamma) \vdash R[\sigma] = R[\tau] : C[\sigma] = C[\tau]$. Second, we show $\Gamma, y\colon 1 \vdash x[\sigma] = M : A[\sigma]$. Note that the first point implies $A[\tau] = A[\sigma]$. By assumption, we have $\Gamma \vdash x[\sigma] = x[\tau][M/x][\text{unit}/y] : A[\sigma]$. Lemma 5.5.6 gives $\Gamma, y\colon 1, x\colon A \vdash x[\tau] = x : A$, so that, by (SUBST-TM-CGR) and transitivity, we can derive $\Gamma \vdash x[\sigma] = M[\text{unit}/y] : A[\sigma]$. Using (1-EQ) and (SUBST-TM-CGR), we get $\Gamma, y\colon 1 \vdash M[\text{unit}/y] = M : A[\sigma]$, which allows us to conclude $\Gamma, y\colon 1 \vdash x[\sigma] = M : A[\sigma]$, as required. Hence, we have shown that $\sigma^\sharp = (\sigma, x[\sigma])$ is the unique map of the required form. □

The above Lemma shows that the fibration $p\colon \mathbb{E} \to \mathbb{B}$ admits comprehension. In [56, 10.4] it is shown that a fibration with comprehension induces a comprehension category, i.e. a functor $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^\to$ that satisfies $p = cod \circ \mathscr{P}$ and that maps cartesian morphisms to pullback squares. This functor is defined by mapping an object B in $\mathbb{E}_\Gamma$ to the map $p(\varepsilon_B)\colon p1(\{B\}) = \{B\} \to \Gamma$, where ε_B is the counit of the adjunction $1 \dashv \{-\}$. In the present case, $\mathscr{P}(B)$ can be described as the weakening $\langle\rangle\colon (\Gamma, x\colon B) \to \Gamma$.

Lemma 7.1.9. *The split comprehension category with unit $p \dashv 1 \dashv \{-\}$ has split products.*

Proof. For an object $(\Gamma \vdash A \text{ Type})$ in $\mathbb{E}_\Gamma$, define the functor $\Pi_A\colon \mathbb{E}_{\{A\}} \to \mathbb{E}_\Gamma$ as follows. An object $\Gamma, x\colon A \vdash B$ Type in $\mathbb{E}_{\{A\}}$ is mapped to $\Gamma \vdash \Pi x\colon A.B$ Type and a morphism $(id, (\Gamma, x\colon A, y\colon C \vdash M : B))$ is mapped to the morphism $(id, (\Gamma, z\colon \Pi x\colon A.C \vdash \lambda x\colon A.M[\text{app}_{(x:A)C}(z,x)/y] : \Pi x\colon A.B))$. Functoriality follows using the β and η-equations.

For each object $(\Gamma \vdash A \text{ Type})$ in $\mathbb{E}_\Gamma$, an adjunction $\pi_A^* \dashv \Pi_A$ follows by a standard argument. It remains to show the Beck-Chevalley condition. It suffices to show that, for all $\sigma\colon \Gamma \to \Delta$ in $\mathbb{B}$, all objects $(\Delta \vdash A \text{ Type})$ in $\mathbb{E}_\Delta$ and all objects $(\Delta, x\colon A \vdash B \text{ Type})$ in $\mathbb{E}_{\{A\}}$, we have $\sigma^*\Pi_A B = \Pi_{\sigma^*A}\{\overline{\sigma}\}^*B$ and $\sigma^*\varepsilon_{A,B} = \varepsilon_{\sigma^*A,\{\overline{\sigma}\}^*B}$, where ε_A is the counit of the adjunction $\pi_A^* \dashv \Pi_A$.

For $\sigma^*\Pi_A B = \Pi_{\sigma^*A}\{\overline{\sigma}\}^*B$, we have to show

$$\Gamma \vdash (\Pi x\colon A.B)[\sigma] = (\Pi y\colon A[\sigma].B[\sigma][y/x])\ \mathrm{Type},$$

where y is fresh. Notice that σ may contain let-terms, so that the equation involves commuting conversions. However, we do not need to derive the commuting conversion, since $(\text{let } M \text{ be } x{*}y \text{ in } \Pi x\colon A.B)$ is defined to be $\Pi x\colon (\text{let } M \text{ be } x{*}y \text{ in } A).(\text{let } M \text{ be } x{*}y \text{ in } B)$, using which the equation becomes immediate.

For showing $\sigma^*\varepsilon_{A,B} = \varepsilon_{\sigma^*A,\{\overline{\sigma}\}^*B}$, we note that the vertical counit $\varepsilon_{A,B}\colon (\Gamma, x\colon A \vdash \Pi x\colon A.B\ \mathrm{Type}) \to (\Gamma, x\colon A \vdash B\ \mathrm{Type})$ is given by $\Delta, x\colon A, y\colon \Pi x\colon A.B \vdash \mathrm{app}_{(x:A)B}(y,x) : B$, where we can assume x and y to be fresh for σ. Hence, we need to show

$$\Gamma, x\colon A[\sigma], y\colon (\Pi x\colon A.B)[\sigma] \vdash (\mathrm{app}_{(x:A)B}(y,x))[\sigma] = \mathrm{app}_{(x:A[\sigma])B[\sigma]}(y,x) : B[\sigma]$$

As an aside, we remark that the normal way of stating the Beck-Chevalley condition does, of course, follow from that equation:

$$\begin{aligned}
\mathrm{app}_{(x:A)B}(M,N)[\sigma] &= \mathrm{app}_{(x:A)B}(y,x)[\langle M/y\rangle \circ \langle N/x\rangle \circ \sigma] \\
&= \mathrm{app}_{(x:A)B}(y,x)[\langle M/y\rangle \circ \sigma \circ \langle N[\sigma]/x\rangle] \\
&= \mathrm{app}_{(x:A)B}(y,x)[\sigma \circ \langle M[\sigma]/y\rangle \circ \langle N[\sigma]/x\rangle] \\
&= \mathrm{app}_{(x:A[\sigma])B[\sigma]}(y,x)[\langle M[\sigma]/y\rangle \circ \langle N[\sigma]/x\rangle] \\
&= \mathrm{app}_{(x:A[\sigma])B[\sigma]}(M[\sigma],N[\sigma])
\end{aligned}$$

The proof goes by induction on the substitution σ. The cases for composition and normal substitution $\langle M/x\rangle$ are straightforward. The interesting case is where σ has the form $\langle M/u\colon C * v\colon D\rangle$. As we observed in the proof of Lemma 5.5.5, in this case there exist derivable judgements $\vdash \Gamma = \Gamma''$ Bunch, $\Gamma'' \succcurlyeq (\Phi_\circ[\langle N/u * v\rangle])(\Gamma')$ and $\Phi(u\colon C * v\colon D) \succcurlyeq \Delta$, and that we have $\Gamma' \vdash N : C{*}D$. We can therefore derive

$$(*\text{-E})\ \frac{(\textsc{Cong})\ \dfrac{\Delta, x\colon A \vdash B\ \mathrm{Type}}{\Phi(u\colon C * v\colon D), x\colon A \vdash B\ \mathrm{Type}}}{(\Phi_\circ[\langle z/u\colon C * v\colon D\rangle])(z\colon C{*}D), x\colon (\text{let } z \text{ be } u{*}v \text{ in } A) \vdash \text{let } z \text{ be } u{*}v \text{ in } B\ \mathrm{Type}}$$

Writing $\Phi'_\circ$ for $(\Phi_\circ[\langle z/u\colon C * v\colon D\rangle])$ and A' for $(\text{let } z \text{ be } u{*}v \text{ in } A)$ and B' for $(\text{let } z \text{ be } u{*}v \text{ in } B)$, we can use this to derive

$$\Phi'(z\colon C{*}D), x\colon A', y\colon \Pi x\colon A'.B' \vdash \mathrm{app}_{(x:A')B'}(y,x) : B'.$$

Using the rules $(*\text{-}\eta)$ and $(*\text{-}\beta)$, and by noticing $A'[M/z] = A[\sigma]$ and $B'[M/z] = B[\sigma]$, we obtain

$$\Phi'[M/z](\Gamma'), x\colon A[\sigma], y\colon \Pi x\colon A[\sigma].B[\sigma] \vdash \mathrm{app}_{(x:A[\sigma])B[\sigma]}(y,x) = \text{let } M \text{ be } u{*}v \text{ in } \mathrm{app}_{(x:A)B}(y,x) : B[\sigma]$$

Because of $\Gamma'' \succcurlyeq (\Phi_\circ[\langle M/u\colon C * v\colon D\rangle])(\Gamma')$, we can use (Cong) to replace $\Phi'[M/z](\Gamma')$ by Γ'' in this sequent. Because of $\vdash \Gamma = \Gamma''$ Bunch, we can further replace Γ'' by Γ. Since the right-hand side of the equation is just $\mathrm{app}_{(x:A)B}(y,x)[\sigma]$, the case is completed by using symmetry of equality. □

Lemma 7.1.10. *The split comprehension category with unit $p \dashv 1 \dashv \{-\}$ has split strong sums.*

Proof. For an object $(\Gamma \vdash A \text{ Type})$ in $\mathbb{E}_\Gamma$, define a functor $\Sigma_A : \mathbb{E}_{\{A\}} \to \mathbb{E}_\Gamma$ as follows. An object $\Gamma, x\colon A \vdash B$ Type of $\mathbb{E}_{\{A\}}$ is mapped to $\Gamma \vdash \Sigma x\colon A.B$ Type and a morphism $(id, (\Gamma, x\colon A, y\colon C \vdash M : B))$ is mapped to $(id, (\Gamma, z\colon \Sigma x\colon A.C \vdash \mathrm{pair}_{(x:A)B}(\mathrm{fst}_{(x:A)C}(z), M[\mathrm{fst}_{(x:A)C}(z)/x][\mathrm{snd}_{(x:A)C}(z)/y]) : \Sigma x\colon A.B))$. Functoriality follows using the β and η-equations.

For each object $(\Gamma \vdash A \text{ Type})$ in $\mathbb{E}_\Gamma$, an adjunction $\Sigma_A \dashv \pi_A^*$ is easily verified, and the Beck-Chevalley condition follows as in the previous lemma. Finally, the sums are strong because the β and η-equations for Σ-types make the following two maps mutually inverse.

$$\langle \mathrm{pair}_{(x:A)B}(x,y)/z\rangle\colon (\Gamma, x\colon A, y\colon B) \to (\Gamma, z\colon \Sigma x\colon A.B)$$
$$\langle \mathrm{fst}_{(x:A)B}(z)/x\rangle \circ \langle \mathrm{snd}_{(x:A)B}(z)/x\rangle\colon (\Gamma, z\colon \Sigma x\colon A.B) \to (\Gamma, x\colon A, y\colon B)$$

□

Lemma 7.1.11. *The split comprehension category with unit $p \dashv 1 \dashv \{-\}$ is a split closed comprehension category.*

Proof. It has products and strong sums by the previous two lemmas. The base category $\mathbb{B}$ has a terminal object, namely the empty context. It remains to show that the comprehension functor $\mathscr{P}\colon \mathbb{E} \to \mathbb{B}^{\to}$, as defined above, is full and faithful. To show that $\mathscr{P}$ is full, consider a morphism in $\mathbb{B}^{\to}$ between objects that are in the image of $\mathscr{P}$, as given by the following commuting square.

$$\begin{array}{ccc} (\Gamma, x\colon A) & \xrightarrow{\tau} & (\Delta, y\colon B) \\ {\scriptstyle\langle\rangle}\downarrow & & \downarrow{\scriptstyle\langle\rangle} \\ \Gamma & \xrightarrow[\sigma]{} & \Delta \end{array}$$

We have to show that this morphism is in the image of the comprehension functor $\{-\}$. By the identification of morphisms up to bijective renaming, we can assume both x and y to be fresh for Γ, Δ and σ. By definition of $\{-\}$, it suffices to show $\tau = \sigma \circ \langle M/y\rangle\colon (\Gamma, x\colon A) \to (\Delta, y\colon B)$ for some term $\Gamma, x\colon A \vdash M : B[\sigma]$. Because the square commutes, we have $\Gamma, x\colon A \vdash B[\sigma] = B[\tau]$ Type. Hence, we get $\Gamma, x\colon A \vdash y[\tau] : B[\sigma]$. Take $M \stackrel{\text{def}}{=} y[\tau]$. Lemma 5.5.9 gives $\tau = \langle z/y\rangle \circ \sigma \circ \langle y[\tau]/z\rangle\colon (\Gamma, x\colon A) \to (\Delta, y\colon B)$ for some/any fresh variable z. It is straightforward to show by induction on σ that $\sigma = \langle z/y\rangle \circ \sigma \circ \langle y/z\rangle\colon (\Gamma, y\colon B[\sigma]) \to (\Delta, y\colon B)$ holds, since both y and z are fresh for σ. This gives $\sigma \circ \langle y[\tau]/y\rangle = \langle z/y\rangle \circ \sigma \circ \langle y/z\rangle \circ \langle y[\tau]/y\rangle\colon (\Gamma, x\colon A) \to (\Delta, y\colon B)$ and thus $\sigma \circ \langle y[\tau]/y\rangle = \langle z/y\rangle \circ \sigma \circ \langle y[\tau]/z\rangle\colon (\Gamma, x\colon A) \to (\Delta, y\colon B)$. Therefore, we have $\tau = \sigma \circ \langle y[\tau]/y\rangle\colon (\Gamma, x\colon A) \to (\Delta, y\colon A)$, as required.

For faithfulness of $\mathscr{P}$, suppose that $(\sigma, (\Gamma, x\colon A \vdash M : B[\sigma]))\colon (\Gamma \vdash A \text{ Type}) \to (\Delta \vdash B \text{ Type})$ and $(\tau, (\Gamma, x\colon A \vdash N : B[\tau]))\colon (\Gamma \vdash A \text{ Type}) \to (\Delta \vdash B \text{ Type})$ are mapped by $\mathscr{P}$ to the same morphism, which means that we have $\sigma = \tau\colon \Gamma \to \Delta$ and $\sigma \circ \langle M/y\rangle = \tau \circ \langle N/y\rangle\colon (\Gamma, x\colon A) \to (\Delta, y\colon B)$ for some/any

fresh variable y. To show $(\sigma, M) = (\tau, N)$, it suffices to show $\Gamma, x\colon A \vdash M = N : B[\sigma] = B[\tau]$. But from $\sigma \circ \langle M/y\rangle = \tau \circ \langle N/y\rangle \colon (\Gamma, x\colon A) \to (\Delta, y\colon B)$ we obtain $\Gamma, x\colon A \vdash y[\sigma \circ \langle M/y\rangle] = y[\tau \circ \langle N/y\rangle] : B[\sigma] = B[\tau]$, making use of the fact that y is not free in $B[\sigma]$ or $B[\tau]$. The equality $M = N$ then follows using Lemma 5.5.6. □

Lemma 7.1.12. *The split comprehension category with unit $p \dashv 1 \dashv \{-\}$ has strong Π^*-types.*

Proof. For objects Γ and A of $\mathbb{B}$, define the functor $\Pi^*_A \colon \mathbb{E}_{\Gamma * (x\colon A)} \to \mathbb{E}_\Gamma$ as follows. An object $\Gamma * x\colon A \vdash B$ Type in $\mathbb{E}_{\Gamma*(x\colon A)}$ is mapped to $\Gamma \vdash \Pi^* x\colon A.B$ Type and a morphism $(id, ((\Gamma * x\colon A), y\colon C \vdash M : B))$ is mapped to the morphism $(id, (\Gamma, z\colon \Pi^* x\colon A.C \vdash \lambda^* x\colon A.M[\mathrm{app}^*_{(x:A)C}(z,x)/y] : \Pi^* x\colon A.B))$. This term is derivable by the following derivation

$$\dfrac{\dfrac{\dfrac{(\Gamma * x\colon A), y\colon C \vdash M : B}{((\Gamma, z\colon \Pi^* x\colon A.C) * x\colon A), y\colon C \vdash M : B}\,(\textsc{Weak})}{(\Gamma, z\colon \Pi^* x\colon A.C) * x\colon A \vdash M[\mathrm{app}^*_{(x:A)C}(z,x)/y] : B}\,(\textsc{Subst})}{\Gamma, z\colon \Pi^* x\colon A.C \vdash \lambda^* x\colon A.M[\mathrm{app}^*_{(x:A)C}(z,x)/y] : \Pi^* x\colon A.B}\,(\Pi^*\text{-I})$$

In the inference step for substitution, we use that $y \notin FV(B)$ holds and that the following is derivable.

$$\dfrac{\Gamma, z\colon \Pi^* x\colon A.C \vdash z : \Pi^* x\colon A.C \qquad x\colon A \vdash x : A}{(\Gamma, z\colon \Pi^* x\colon A.C) * x\colon A \vdash \mathrm{app}^*_{(x:A)C}(z,x) : B}$$

Functoriality follows using the β and η-equations.

To show that this defines strong Π^*-types, we have to give a natural isomorphism between $\mathbb{E}_\Gamma(B, \Pi^*_A C)$ and $\mathbb{B}/(\Gamma * A)(\pi_B * A, \pi_C)$ and prove the stronger Beck-Chevalley condition from Definition 2.5.12. To give the natural isomorphism, it suffices by Lemma 2.5.8 to give, for all Γ and A in $\mathbb{B}$ and all B in $\mathbb{E}_{\Gamma*(x\colon A)}$, a morphism $\varepsilon_{A,B} \colon \pi_{\Pi^*_A B} * (x\colon A) \to \pi_B$ in $\mathbb{B}/(\Gamma * x\colon A)$, such that $\varepsilon_{A,B}$ is natural in B and, for each morphism $m\colon \pi_C * (x\colon A) \to \pi_B$ in $\mathbb{B}/(\Gamma * x\colon A)$, there exists a unique morphism $\lambda^* m\colon C \to \Pi^*_A B$ in $\mathbb{E}_\Gamma$ satisfying $\varepsilon_{A,B} \circ (\{\lambda^* m\} * (x\colon A)) = m$. Define

$$\varepsilon_{A,B} \stackrel{\text{def}}{=} \langle \mathrm{app}^*_{(x:A)B}(y,x)/z\rangle \colon (\Gamma, y\colon \Pi^* x\colon A.B) * x\colon A \to (\Gamma * x\colon A), z\colon B.$$

Note that $\pi_{\Pi^*_A B} \colon (\Gamma, y\colon \Pi^* x\colon A.B) \to \Gamma$ and $\pi_B \colon (\Gamma * x\colon A), z\colon B \to \Gamma * x\colon A$ are both given by $\langle\rangle$, so that $\varepsilon_{A,B}$ becomes a morphism in $\mathbb{B}/(\Gamma * x\colon A)$.

We verify naturality of $\varepsilon_{A,B}$. Let $\sigma \colon (\Gamma \vdash C \text{ Type}) \to (\Gamma \vdash B \text{ Type})$ be a morphism in $\mathbb{E}_\Gamma$. By definition of $\mathbb{E}$, this morphism is uniquely determined by a term $\Gamma, u\colon C \vdash M : B$. For naturality of $\varepsilon_{A,B}$, it then suffices to show $\langle M/z\rangle \circ \langle \mathrm{app}^*_{(x:A)C}(v,x)/u\rangle = \langle \mathrm{app}^*_{(x:A)B}(y,x)/z\rangle \circ \langle \lambda^* x\colon A.M[\mathrm{app}^*_{(x:A)C}(v,x)/u]/y\rangle$ of type $(\Gamma, v\colon \Pi^* x\colon A.C) * x\colon A \to (\Gamma * x\colon A), z\colon B$. Since u and y are fresh for the codomain of these two morphisms, it follows immediately that the first morphisms equals $\langle M[\mathrm{app}^*_{(x:A)C}(v,x)/u]/z\rangle$ and second morphism equals $\langle \mathrm{app}^*_{(x:A)B}((\lambda^* x\colon A.M[\mathrm{app}^*_{(x:A)C}(v,x)/u]), x)/z\rangle$, both of the same type. It follows using β-equality that both substitutions are equal.

Finally, the strong Beck-Chevalley condition is given exactly by the rule (Π^*-E-CGR$^+$). □

Lemma 7.1.13. *For any two objects* $(\vdash A \text{ Type})$ *and* $(\vdash B \text{ Type})$ *in* $\mathbb{E}$ *over the empty context, the map* $\langle \text{pair}^*_{A,B}(x,y)/z\rangle \colon \{\vdash A \text{ Type}\} * \{\vdash B \text{ Type}\} = ((x\colon A) * (y\colon B)) \to (z\colon A*B) = \{\vdash A*B \text{ Type}\}$ *is an isomorphism.*

Proof. Its inverse is given by $\langle z/x\colon A * y\colon B\rangle \colon (z\colon A*B) \to (x\colon A) * (y\colon B)$. □

Lemma 7.1.14. *The affine symmetric monoidal structure* $*$ *on* $\mathbb{B}$ *is a strict affine monoidal structure.*

Proof. Let Γ and Δ be an objects of $\mathbb{B}$. The canonical projections out of $\Gamma * \Delta$ are given by weakenings $\langle\rangle \colon \Gamma * \Delta \to \Gamma$ and $\langle\rangle \colon \Gamma * \Delta \to \Delta$. We have to show that these two maps are jointly monic. Because of the isomorphisms $(x\colon A, y\colon B) \cong (z\colon \Sigma^* x\colon A.B)$ and $(x\colon A) * (y\colon B) \cong (z\colon A*B)$, we can assume that both Γ and Δ consist of one declaration only, say Γ is $(u\colon C)$ and Δ is $(v\colon D)$.

Because $\mathscr{P}$ is full and faithful, as shown in Lemma 7.1.11, any morphism $\sigma \colon \Phi \to (x\colon A)$ corresponds uniquely to the term $\Phi \vdash x[\sigma] : A$. With the isomorphism $\langle \text{pair}^*_{C,D}(u,v)/z\rangle \colon (u\colon C) * (v\colon D) \to (z\colon C*D)$, this implies that a morphism $\sigma \colon \Phi \to (u\colon C) * (v\colon D)$ corresponds uniquely to the term $\Phi \vdash (\text{pair}^*_{C,D}(u,v))[\sigma] : C*D$. Likewise, $\dot{\pi}_1 \circ \sigma$ and $\dot{\pi}_2 \circ \sigma$ correspond to the terms $\Phi \vdash u[\sigma] : C$ and $\Phi \vdash v[\sigma] : D$ respectively. Therefore, to show that the projections are jointly monic, it suffices to show that $\Phi \vdash (\text{pair}^*_{C,D}(u,v))[\sigma] = (\text{pair}^*_{C,D}(u,v))[\tau] : C*D$ holds for any two substitutions σ and τ of type $\Phi \to (u\colon C) * (v\colon D)$ that satisfy $\Phi \vdash u[\sigma] = u[\tau] : C$ and $\Phi \vdash v[\sigma] = v[\tau] : D$. By the rule (INJECT), it suffices to show the two equations

$$\Phi \vdash \text{let } (\text{pair}^*_{C,D}(u,v))[\sigma] \text{ be } u{:}C*v{:}D \text{ in } u = \text{let } (\text{pair}^*_{C,D}(u,v))[\tau] \text{ be } u{:}C*v{:}D \text{ in } u : C$$

$$\Phi \vdash \text{let } (\text{pair}^*_{C,D}(u,v))[\sigma] \text{ be } u{:}C*v{:}D \text{ in } v = \text{let } (\text{pair}^*_{C,D}(u,v))[\tau] \text{ be } u{:}C*v{:}D \text{ in } v : D.$$

By straightforward induction on σ, we can derive the following equation.

$$\Phi \vdash \text{let } (\text{pair}^*_{C,D}(u,v))[\sigma] \text{ be } u{:}C*v{:}D \text{ in } u = (\text{let } \text{pair}^*_{C,D}(u,v) \text{ be } u{:}C*v{:}D \text{ in } u)[\sigma] = u[\sigma] : C$$

Corresponding equations hold for the other let-terms. Hence, the assumptions $\Phi \vdash u[\sigma] = u[\tau] : C$ and $\Phi \vdash v[\sigma] = v[\tau] : D$ suffice to prove the required $\Phi \vdash (\text{pair}^*_{C,D}(u,v))[\sigma] = (\text{pair}^*_{C,D}(u,v))[\tau] : C*D$. □

Putting all the above lemmas together, we obtain:

Proposition 7.1.15. *The syntax of* $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ *defines a* $(*,1,\Sigma,\Pi,\Pi^*)$*-type category.*

Finally, we obtain the following completeness result.

Proposition 7.1.16 (Completeness)**.** *For any dependently typed algebraic theory* $\mathscr{T}$ *in* $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$*, the following two statements hold.*

- *If both* $\Gamma \vdash A$ Type *and* $\Gamma \vdash B$ Type *are derivable in* $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ *and have the same interpretation in all* $(*,1,\Sigma,\Pi,\Pi^*)$*-models for* $\mathscr{T}$*, then* $\Gamma \vdash A = B$ Type *is derivable.*
- *If both* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : A$ *are derivable in* $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ *and have the same interpretation in all* $(*,1,\Sigma,\Pi,\Pi^*)$*-models for* $\mathscr{T}$*, then* $\Gamma \vdash M = N : A$ *is derivable.*

We are being slightly imprecise in the statement of this proposition, since the notion of a $(*, 1, \Sigma, \Pi, \Pi^*)$-model has, strictly speaking, been defined only for theories in **ES**. It should be clear, however, how to use the interpretation from the previous chapter to extend the definition of a model to theories in **BT**.

Proof. Using Proposition 7.1.15 we obtain that the syntax of $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$ forms a $(*, 1, \Sigma, \Pi, \Pi^*)$-model for $\mathscr{T}$. Now, if two terms (respectively types) receive the same interpretation in all models for $\mathscr{T}$, they must in particular have the same interpretation in the term model. The derivability of the asserted equations follows from this by the definition of the interpretation in the term-model. □

We remark that a stronger form of completeness is likely to be true, namely that there is an equivalence between categories of **BT**-theories and categories of $(*, 1, \Sigma, \Pi, \Pi^*)$-models, as in e.g. [99]. However, due to type-dependency, verifying all the details becomes rather involved. Since, at present, we do not have an application for such a stronger completeness result, we confine ourselves to the weaker above.

7.2 Consequences of Completeness

In this section we consider some consequences of completeness. We observe that the type of each term is unique up to provable equality. Moreover, we show that any two terms differing only up to commuting conversion of $*$-lets are provably equal. These direct consequences of completeness suggest ways of dealing with some of the problems described earlier. In particular, in Section 7.2.3, we will see how substitution can be shown to be admissible up to commuting conversion.

7.2.1 Interpretation in the Term Model

We start by spelling out the interpretation of $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$ in the term model. This interpretation is essentially the identity, as formulated in the following lemma. It should be clear how, from a dependently typed algebraic theory $\mathscr{T}_{\mathbf{BT}}$ in $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$, we can define a compatible algebraic theory $\mathscr{T}_{\mathbf{ES}}$ in **ES**, when **ES** is intended to be modelled in the term model constructed above.

Lemma 7.2.1.

- *If* $\vdash_{\mathbf{BT}} \Gamma$ Bunch *then there exists a derivation of* $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch*, such that the interpretation of* $\vdash_{\mathbf{ES}} \Gamma'$ Bunch *in the term model is the object of* $\mathbb{B}$ *given by* $\vdash_{\mathbf{BT}} \Gamma$ Bunch*.*
- *If* $\Gamma \vdash_{\mathbf{BT}} A$ Type *then there exist derivations of* $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch *and* $\Gamma' \vdash_{\mathbf{ES}} A' \Downarrow |A|$ Type*, such that the interpretation of* $\Gamma' \vdash_{\mathbf{ES}} A'$ Type *in the term model is the object of* $\mathbb{E}_\Gamma$ *given by* $\Gamma \vdash_{\mathbf{BT}} A$ Type*.*
- *If* $\Gamma \vdash_{\mathbf{BT}} M : A$ *then there exist derivations of* $\vdash_{\mathbf{ES}} \Gamma' \Downarrow |\Gamma|$ Bunch *and* $\Gamma' \vdash_{\mathbf{ES}} A' \Downarrow |A|$ Type *and* $\Gamma' \vdash_{\mathbf{ES}} M' \Downarrow |M| : A'$*, such that the interpretation of* $\Gamma' \vdash_{\mathbf{ES}} M' : A'$ *in the term model is the morphism of* $\mathbb{E}_\Gamma$ *given by the pair* $(id, (\Gamma, u\colon 1 \vdash_{\mathbf{BT}} M : A))$*, where u is a fresh variable.*

Proof. The existence of derivations in **ES** follows just as in the proof of Proposition 6.5.33. By examining that proof, it follows that the judgements in **ES** constructed there have an interpretations in the term model that differs from the original judgement only up to bijective renaming of variables. The assertion then follows, since the objects and morphisms in $\mathbb{B}$ and $\mathbb{E}$ are identified up to bijective renaming. □

Corollary 7.2.2.

- *The interpretation of* $\vdash_{\mathbf{BT}} \Gamma$ Bunch *in the term model is the object of* $\mathbb{B}$ *given by* $\vdash_{\mathbf{BT}} \Gamma$ Bunch.
- *The interpretation of* $\Gamma \vdash_{\mathbf{BT}} A$ Type *in the term model is the object of* $\mathbb{E}_\Gamma$ *given by* $\Gamma \vdash_{\mathbf{BT}} A$ Type.
- *The interpretation of* $\Gamma \vdash_{\mathbf{BT}} M : A$ *in the term model is the morphism of* $\mathbb{E}_\Gamma$ *given by the pair* $(id, (\Gamma, u\colon 1 \vdash_{\mathbf{BT}} M : A))$*, where* u *is a fresh variable.*

7.2.2 Commuting Conversion

Having observed the interpretation in the term model to be essentially the identity, we can now show that any two terms (respectively types) differing only up to commuting conversion are provably equal. By commuting conversion, we mean equations such as

$$(\text{let } R \text{ be } x{*}y \text{ in } \text{app}(M,N)) = \text{app}(\text{let } R \text{ be } x{*}y \text{ in } M, \text{let } R \text{ be } x{*}y \text{ in } N).$$

Such equations are most economically expressed using the translation to the let-free syntax $|-|$ from Chapter 6. We say that two terms (respectively types) M and N are equal up to commuting conversion if $|M|$ and $|N|$ are syntactically equal.

Proposition 7.2.3. *The following hold in* $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$.

1. *If* $\Gamma \vdash A$ Type *and* $\Gamma \vdash B$ Type *and* $|A| = |B|$ *then* $\Gamma \vdash A = B$ Type.
2. *If* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : B$ *and* $|M| = |N|$ *then* $\Gamma \vdash A = B$ Type *and* $\Gamma \vdash M = N : A$.

Proof. We consider the second point, the first point follows similarly. From Lemma 7.2.1, we obtain judgements $\Gamma' \Downarrow |\Gamma| \vdash_{\mathbf{ES}} M' \Downarrow |M| : A' \Downarrow |A|$ and $\Gamma'' \Downarrow |\Gamma| \vdash_{\mathbf{ES}} N' \Downarrow |N| : B' \Downarrow |B|$ whose interpretations in the term model are given by $(id, (\Gamma, u\colon 1 \vdash_{\mathbf{BT}} M : A))$ and $(id, (\Gamma, u\colon 1 \vdash_{\mathbf{BT}} N : B))$ respectively, where u is a fresh variable. Using Lemma 6.5.27 we get $\vdash_{\mathbf{ES}} \Gamma' = \Gamma''$ Bunch, which by (BU-CONV) implies $\Gamma' \vdash_{\mathbf{ES}} N' : B'$. Since, by assumption, we have $|M| = |N|$, we can use Lemma 6.5.26 to obtain $\Gamma' \vdash_{\mathbf{ES}} A' = B'$ Type and $\Gamma' \vdash_{\mathbf{ES}} M' = N' : A'$. By soundness of the interpretation (Proposition 6.4.13) in the term model, this implies that A' and B' as well as M' and N' have the same interpretation. Hence, we have $\Gamma \vdash_{\mathbf{BT}} A = B$ Type and $\Gamma, u\colon 1 \vdash_{\mathbf{BT}} M = N : A$. Since u is fresh, substituting $(\text{unit}\colon 1)$ for u gives the required $\Gamma \vdash_{\mathbf{BT}} M = N : A$. □

Corollary 7.2.4. *If* $\Gamma \vdash M : A$ *and* $\Gamma \vdash M : B$ *then* $\Gamma \vdash A = B$ Type.

7.2.3 Admissibility of Substitution

In the discussion of the bunched type theory with $*$-types in Chapter 5, we have noted that the presence of parameter contexts in rule ($*$-E) makes the substitution rule (SUBST) non-admissible. In this section we show how (SUBST) can be made admissible up to commuting conversion. We do this by showing $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ to be equivalent to a type theory $\mathbf{BT}^-$ in which the rule ($*$-E) is replaced by a rule ($*$-E$'$) without parameter contexts, and for which substitution is admissible. Although this is not strictly speaking a consequence of completeness, the definition of $\mathbf{BT}^-$ is suggested by Proposition 7.2.3.

The type theory $\mathbf{BT}^-$ is defined as $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*)$ without the rules (SUBST), ($*$-E), ($*$-E-CGR), ($*$-β) and ($*$-η), but with the following rules instead.

$$(*\text{-E}')\ \frac{\Delta \vdash M : A * B \qquad \Gamma * x\colon A * y\colon B \vdash \mathscr{J}}{\Gamma * \Delta \vdash \text{let } M \text{ be } x{:}A * y{:}B \text{ in } \mathscr{J}}$$

$$(*\text{-E}'\text{-CGR})\ \frac{\begin{array}{cc} \vdash A_1 = A_2 \text{ Type} & \Delta \vdash M_1 = M_2 : A_1 * B_1 \\ \vdash B_1 = B_2 \text{ Type} & \Gamma * x\colon A_1 * y\colon B_1 \vdash \mathscr{J} \end{array}}{\Gamma * \Delta \vdash \text{let } M_1 \text{ be } x{:}A_1 * y{:}B_1 \text{ in } \mathscr{J} = \text{let } M_2 \text{ be } x{:}A_2 * y{:}B_2 \text{ in } \mathscr{J}}$$

$$(*\text{-}\beta')\ \frac{\Gamma \vdash M : A \qquad \Delta \vdash N : B \qquad \Phi * x\colon A * y\colon B \vdash \mathscr{K}}{\Phi * \Gamma * \Delta \vdash (\text{let } \text{pair}^*_{A,B}(M,N) \text{ be } x{:}A * y{:}B \text{ in } \mathscr{K}) = (\mathscr{K}\,[M/x]\,[N/y])}$$

$$(*\text{-}\eta')\ \frac{\Delta \vdash M : A * B \qquad \Gamma * z\colon A * B \vdash \mathscr{K}}{\Gamma * \Delta \vdash \mathscr{K}\,[M/z] = \text{let } M \text{ be } x{:}A * y{:}B \text{ in } (\mathscr{K}\,[\text{pair}^*_{A,B}(x,y)/z])}$$

$$(\text{BU-CC})\ \frac{\vdash \Gamma \text{ Bunch} \qquad \vdash \Delta \text{ Bunch} \qquad |\Gamma| = |\Delta|}{\vdash \Gamma = \Delta \text{ Bunch}}$$

$$(\text{TY-CC})\ \frac{\Gamma \vdash A \text{ Type} \qquad \Gamma \vdash B \text{ Type} \qquad |A| = |B|}{\Gamma \vdash A = B \text{ Type}}$$

$$(\text{TM-CC})\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : A \qquad |M| = |N|}{\Gamma \vdash M = N : A}$$

In these rules, $\mathscr{J}$ ranges over arbitrary judgements, as defined in Chapter 4, and $\mathscr{K}$ ranges over $(A \text{ Type})$ and $(M\colon A)$. The above rules for $*$-types differ from the rules in Chapter 5 mainly in that they do not contain parameter contexts. Because $\mathscr{K}$ ranges over both types and terms, the above rules include both let on terms and types. Thus, the let on types is now a primitive notion and not defined as in **BT**. The basic properties established in Chapter 4 continue to hold for $\mathbf{BT}^-$.

We have already shown in Section 5.4 and Proposition 7.2.3 that all the rules of $\mathbf{BT}^-$ are admissible in **BT**. It may be the case that the rules (TY-CC) and (TM-CC) are also admissible in $\mathbf{BT}^-$. However, Proposition 7.2.3 does not imply this, since its proof takes place in **BT** and may therefore

make use of rules such as (SUBST) and ($*$-E), which are not available in $\mathbf{BT}^-$. Likewise, the argument of Section 5.4, showing that let on types is admissible in **BT**, does not apply to $\mathbf{BT}^-$. For instance, the assumptions on the contexts in ($*$-E$'$) prevent us from transforming $(\text{let } z \text{ be } x*y \text{ in } \Pi u\colon A.B)$ into $\Pi u\colon (\text{let } z \text{ be } x*y \text{ in } A).(\text{let } z \text{ be } x*y \text{ in } B)$, as the latter would not be a valid type.

Clearly, any derivation in $\mathbf{BT}^-$ can be transformed into a derivation in **BT**. In the rest of this section we show that show any derivation in **BT** can, up to commuting conversion, be transformed into a derivation in $\mathbf{BT}^-$.

Lemma 7.2.5. *The rule* (SUBST) *is admissible for* $\mathbf{BT}^-$.

Proof. By induction on derivations. The case for ($*$-E$'$) is straightforward, since any substitution in $\Gamma*\Delta$ must be either completely inside Γ or inside Δ. In either case, we can use the induction hypothesis. □

Having shown admissibility of substitution, it just remains to show admissibility of the rules of **BT** for $*$-types. These rules are admissible in the following sense.

Definition 7.2.6. A rule

$$\frac{\Gamma_1 \vdash \mathscr{J}_1 \qquad \dots \qquad \Gamma_n \vdash \mathscr{J}_n}{\Gamma \vdash \mathscr{J}}$$

is *admissible up to commuting conversion* if, whenever the premises are derivable, then there exists a derivable sequent $\Gamma' \vdash \mathscr{J}'$ with $|\Gamma'| = |\Gamma|$ and $|\mathscr{J}| = |\mathscr{J}'|$. Here, we extend $|-|$ to judgements $\mathscr{J}$ by $|A \text{ Type}| = (|A| \text{ Type})$ and $|M\colon A| = (|M|\colon |A|)$ and $|A = B \text{ Type}| = (|A| = |B| \text{ Type})$ and $|M = N\colon A| = (|M| = |N|\colon |A|)$.

We start by showing that ($*$-E) is admissible up to commuting conversion. Informally, the idea is simple. We treat the let-terms in ($*$-E) as explicit substitutions and push them inside the structure of the terms until we reach a situation where they can be introduced using just ($*$-E$'$). For example, the term $(\text{let } z \text{ be } x*y \text{ in } \langle x,z\rangle)$, which is not derivable using just ($*$-E$'$), is transformed into $\langle \text{let } z \text{ be } x*y \text{ in } x,z\rangle$, which is derivable using just ($*$-E$'$). Note, however, that this pushing inside of let-terms is not determined by the term alone and may depend on the derivation. For example, if we have $(\text{let } z \text{ be } x*y \text{ in } M*N)$ where neither x not y occur in M or N, then it depends on the derivations of M and N whether we continue with $(\text{let } z \text{ be } x*y \text{ in } M)*N$, with $M*(\text{let } z \text{ be } x*y \text{ in } N)$, or if we leave the term as it is. If we have the strengthening property, then this non-determinacy disappears. Without strengthening, we need the rules (BU-CC), (TY-CC) and (TM-CC) to obtain that the non-determinacy makes no difference.

Lemma 7.2.7. *In* $\mathbf{BT}^-$ *the following rule is admissible up to commuting conversion.*

$$\frac{\Delta \vdash M : A*B \qquad \Gamma(x\colon A * y\colon B) \vdash \mathscr{J}}{(\text{let } M \text{ be } x*y \text{ in } \Gamma_\circ)(\Delta) \vdash \text{let } M \text{ be } x*y \text{ in } \mathscr{J}}$$

In this rule $\mathscr{J}$ *ranges over* C *or* $N\colon C$ *or* $C = D$ *or* $N = R\colon C$.

Proof. We show that the following rules are admissible up to commuting conversion.

$$\frac{x\colon A * y\colon B \succcurlyeq \Delta \qquad \Gamma(\Delta) \equiv \Phi \qquad \vdash \Phi \text{ Bunch}}{\vdash (\text{let } z \text{ be } x{*}y \text{ in } \Gamma_\circ)(z\colon A{*}B) \text{ Bunch}}\, x,y \notin v(\Gamma_\circ)$$

$$\frac{x\colon A * y\colon B \succcurlyeq \Delta \qquad \Gamma(\Delta) \equiv \Phi \qquad \Phi \vdash \mathscr{J}}{(\text{let } z \text{ be } x{*}y \text{ in } \Gamma_\circ)(z\colon A{*}B) \vdash \text{let } z \text{ be } x{*}y \text{ in } \mathscr{J}}\, x,y \notin v(\Gamma_\circ)$$

The assertion of the lemma follows by substitution of M for z.

The proof goes by simultaneous induction on the derivations of $\vdash \Phi$ Bunch and $\Phi \vdash \mathscr{J}$. For either rule, the case where the context Δ is congruent to the empty context is trivial, since weakening with $z\colon A{*}B$ gives a sequent as required. Using this, we can assume that Δ is either $(x\colon A)$ or $(y\colon B)$ or $(x\colon A) * (y\colon B)$. We continue by case distinction on the last rule in a derivation.

- (BU-EMPTY), (BU-ADD), (BU-MULT) are straightforward uses of the induction hypothesis. We do the rule (BU-MULT).

  $$(\text{BU-MULT})\ \frac{\vdash \Phi \text{ Bunch} \qquad \vdash \Psi \text{ Bunch}}{\vdash \Phi * \Psi \text{ Bunch}}$$

  Assume $\Gamma(\Delta) \equiv (\Phi * \Psi)$ with $x\colon A * y\colon B \succcurlyeq \Delta$ and $x,y \notin \Gamma_\circ$. We distinguish two cases. First, all variables of Δ occur in either Φ or Ψ. Without loss of generality, we may assume that the variables of Δ occur in Φ. Then there exists $\Gamma'_\circ$ with $x,y \notin \Gamma'_\circ$ such that $\Gamma'(\Delta) \equiv \Phi$ holds. We can therefore apply the induction hypothesis to the left premise to obtain a sequent $\vdash \Gamma''(z\colon A{*}B)$ Bunch with $|(\text{let } z \text{ be } x{*}y \text{ in } \Gamma'_\circ)(z\colon A{*}B)| = |\Gamma''(z\colon A{*}B)|$. Using rule (BU-MULT), we can derive $\vdash \Gamma''(z\colon A{*}B) * \Psi$ Bunch. Since $\Gamma(\Delta) \equiv \Phi * \Psi \equiv \Gamma'(\Delta) * \Psi$, this context is congruent to a context of the required form. This shows the assertion.

  The second case is where both $v(\Phi) \cap v(\Delta) \neq \emptyset$ and $v(\Psi) \cap v(\Delta) \neq \emptyset$ hold. Without loss of generality we may assume that Δ is $(x\colon A) * (y\colon B)$, that x occurs in Φ and that y occurs in Ψ. Because we have $\Gamma(x\colon A * y\colon B) \equiv \Phi * \Psi$, this implies the existence of Γ' satisfying $\Gamma(x\colon A * y\colon B) \equiv \Gamma' * (x\colon A) * (y\colon B)$. Since $(\text{let } z \text{ be } x{*}y \text{ in } (\Gamma' * \bigcirc))(z\colon A{*}B) = \Gamma' * z\colon A{*}B$, this implies $(\text{let } z \text{ be } x{*}y \text{ in } \Gamma_\circ)(z\colon A{*}B) = \Gamma(z\colon A{*}B)$. Since we have $\vdash \Gamma' * z\colon A{*}B$ Bunch, we also have $\vdash \Gamma(z\colon A{*}B)$ Bunch, which implies the assertion.

- (WEAK).

  $$(\text{WEAK})\ \frac{\Phi'(\Phi'') \vdash \mathscr{J} \qquad \Phi'' \vdash C \text{ Type}}{\Phi'(\Phi'', u\colon C) \vdash \mathscr{J}}$$

  Assume $\Gamma(\Delta) \equiv \Phi'(\Phi'', u\colon C)$ with $x\colon A * y\colon B \succcurlyeq \Delta$ and $x,y \notin \Gamma_\circ$. There are three possible cases. First, all the variables of Δ are declared in Φ''. In this case, we have $\Phi'' \equiv \Psi''(\Delta)$ for some $\Psi''_\circ$ with $x,y \notin \Psi''_\circ$. We can apply the induction hypothesis to get derivations of $\Gamma'(z\colon A{*}B) \vdash \mathscr{J}'$ and $\Gamma''(z\colon A{*}B) \vdash C'$ Type such that we have $|(\text{let } z \text{ be } x{*}y \text{ in } \Phi'(\Psi''_\circ))(z\colon A{*}B)| = |\Gamma'(z\colon A{*}B)|$,

$|\text{let } z \text{ be } x{*}y \text{ in } \mathscr{J}| = |\mathscr{J}'|$, $|(\text{let } z \text{ be } x{*}y \text{ in } \Psi''_\circ)(z\colon A{*}B)| = |\Gamma''(z\colon A{*}B)|$ and $|\text{let } z \text{ be } x{*}y \text{ in } C| = |C'|$. Since $|-|$ preserves the structure of contexts, and by making use of rule (BU-CC), we can show that there exists a context-with-hole $\Gamma'''_\circ$ such that $\vdash \Gamma'(z\colon A{*}B) = \Gamma'''(\Gamma''(z\colon A{*}B))$ Bunch holds. Hence, we can use (WEAK) to make the judgement $\Gamma'''(\Gamma''(z\colon A{*}B), u\colon C') \vdash \mathscr{J}'$, which is as required.

The second case is where all the variables of Δ are declared in Φ'. In this case, we only apply the induction hypothesis to the left premise. We can then use (WEAK) to weaken with $u\colon C$, which yields a sequent with the required property.

The final possible case is where u is either x or y. In this case, the context Φ'' must be structurally congruent to the empty context, so that the sequent we obtain by using the induction hypothesis for the left premise is as required.

- The other structural rules are immediate, since the hypothesis $\Gamma(\Delta) \equiv \Phi$ in the rules includes all structural rules except weakening.

- (SUBST-TY-CGR), (SUBST-TM-CGR). These rules follow from the induction hypothesis by a similar case-distinction as in (WEAK).

- (BU-CONV). This rule follows using the induction hypothesis and ($*$-E$'$-CGR).

- (PROJ).

$$(\text{PROJ})\ \frac{\Phi \vdash C \text{ Type}}{\Phi, u\colon C \vdash u : C}$$

Assume $\Gamma(\Delta) \equiv (\Phi, u\colon C)$ with $x\colon A * y\colon B \succcurlyeq \Delta$ and $x, y \notin \Gamma_\circ$. If u is neither x nor y, then $\Gamma(\Delta)$ is $(\Gamma'(\Delta), u\colon C)$ with $\Gamma'(\Delta) \equiv \Phi$. We can use the induction hypothesis to obtain $\Gamma''(z\colon A{*}B) \vdash C'$ Type with $|(\text{let } z \text{ be } x{*}y \text{ in } \Gamma'_\circ)(z\colon A{*}B)| = |\Gamma''(z\colon A{*}B)|$ and $|\text{let } z \text{ be } x{*}y \text{ in } C| = |C'|$. Then $\Gamma''(z\colon A{*}B), u\colon C' \vdash u : C'$ is derivable and satisfies the assertion. If u is either x or y, without loss of generality $u = x$, then we must have $\Phi \equiv \diamond$, since $\Gamma(x\colon A * y\colon B)$ can be obtained from $(\Phi, x\colon A)$ by weakening and the structural rules, which is possible only if Φ is congruent to the empty context. Since $x, y \notin \Gamma_\circ$ and $\Gamma(\Delta) \equiv (\Phi, x\colon A)$, we furthermore have $\Delta \equiv (x\colon A)$ and $\Gamma_\circ = \bigcirc$. Therefore, the sequent $z\colon A{*}B \vdash \text{let } z \text{ be } x{:}A{*}y{:}B \text{ in } x : A$, which is derivable using ($*$-E$'$), satisfies the assertion.

- (TY-CONV), (TY-EQ-CONV). We consider (TY-CONV).

$$(\text{TY-CONV})\ \frac{\Phi \vdash M : D \qquad \Phi \vdash D = C \text{ Type}}{\Phi \vdash M : C}$$

Assume $\Gamma(\Delta) \equiv \Phi$ with $x\colon A * y\colon B \succcurlyeq \Delta$ and $x, y \notin \Gamma_\circ$. The induction hypothesis for the left premise gives $\Gamma'(z\colon A{*}B) \vdash M' : D'$ with $|(\text{let } z \text{ be } x{*}y \text{ in } \Gamma_\circ)(z\colon A{*}B)| = |\Gamma'(z\colon A{*}B)|$, $|\text{let } z \text{ be } x{*}y \text{ in } M| = |M'|$ and $|\text{let } z \text{ be } x{*}y \text{ in } D| = |D'|$. The other premise yields $\Gamma''(z\colon A{*}B) \vdash D'' = C''$ Type with

$|(\text{let } z \text{ be } x{*}y \text{ in } \Gamma_\circ)(z\colon A{*}B)| = |\Gamma''(z\colon A{*}B)|$, $|\text{let } z \text{ be } x{*}y \text{ in } D| = |D''|$ and $|\text{let } z \text{ be } x{*}y \text{ in } C| = |C''|$. Using rules (BU-CC) and (TY-CC), we get $\Gamma'(z\colon A{*}B) \vdash D' = D'' = C''$ Type, so that we can apply (TY-CONV) to derive $\Gamma'(z\colon A{*}B) \vdash M' : C''$, which is as required.

- (TY-REFL), (TY-SYM), (TY-TRANS), (TM-REFL), (TM-SYM), (TM-TRANS) follow from the induction hypothesis and the commuting conversion rules (BU-CC), (TY-CC) and (TM-CC). We consider (TM-TRANS).

$$(\text{TM-TRANS})\ \frac{\Phi \vdash M = N : C \qquad \Phi \vdash N = P : C}{\Phi \vdash M = P : C}$$

  Assume $\Gamma(\Delta) \equiv \Phi$ with $x\colon A * y\colon B \succcurlyeq \Delta$ and $x, y \notin \Gamma_\circ$. The induction hypothesis gives $\Gamma'(z\colon A{*}B) \vdash M' = N' : C'$ and $\Gamma''(z\colon A{*}B) \vdash N'' = P' : C''$ with $|(\text{let } z \text{ be } x{*}y \text{ in } \Gamma_\circ)(z\colon A{*}B)| = |\Gamma'(z\colon A{*}B)| = |\Gamma''(z\colon A{*}B)|$, $|\text{let } z \text{ be } x{*}y \text{ in } M| = |M'|$, $|\text{let } z \text{ be } x{*}y \text{ in } N| = |N'| = |N''|$, $|\text{let } z \text{ be } x{*}y \text{ in } P| = |P'|$ and $|\text{let } z \text{ be } x{*}y \text{ in } C| = |C'| = |C''|$. We obtain $\Gamma'(z\colon A{*}B) \vdash N'' = P' : C'$ using (BU-CC), (TY-CC) and the conversion rules. By validity, we have $\Gamma'(z\colon A{*}B) \vdash N' : C'$ and $\Gamma'(z\colon A{*}B) \vdash N'' : C'$, from which we obtain $\Gamma'(z\colon A{*}B) \vdash N' = N'' : C'$ by (TM-CC). Using (TM-TRANS), we get $\Gamma'(z\colon A{*}B) \vdash M' = P' : C'$, which is as required.

- (C-TY), (C-TM), (C-TY-CGR), (C-TM-CGR), (TY-EQ-AX), (TM-EQ-AX). These rules follow from the induction hypothesis, using the fact that, for a closed type $\vdash C$ Type, the equation $\Gamma(z\colon A{*}B) \vdash \text{let } z \text{ be } x{*}y \text{ in } C = C$ Type is derivable in $\mathbf{BT}^-$. We consider (C-TM).

$$(\text{C-TM})\ \frac{\Phi \vdash M : C \qquad x\colon C \vdash D \text{ Type}}{\Phi \vdash f(M) : D[M/x]}\ f \in \mathscr{T}(x\colon C; D)$$

  Assume $\Gamma(\Delta) \equiv \Phi$ with $x\colon A * y\colon B \succcurlyeq \Delta$ and $x, y \notin \Gamma_\circ$. From the induction hypothesis, we obtain $\Gamma'(z\colon A{*}B) \vdash M' : C'$ with $|(\text{let } z \text{ be } x{*}y \text{ in } \Gamma_\circ)(z\colon A{*}B)| = |\Gamma'(z\colon A{*}B)|$ and $|\text{let } z \text{ be } x{*}y \text{ in } M| = |M'|$ and $|\text{let } z \text{ be } x{*}y \text{ in } C| = |C'|$. Since $x\colon C \vdash D$ Type is derivable, validity implies that so is $\vdash C$ Type, and by weakening also $\Gamma'(z\colon A{*}B) \vdash C$ Type. Since C is closed, we have $|\text{let } z \text{ be } x{*}y \text{ in } C| = |C|$. Hence, we can use (TY-CC) to get $\Gamma'(z\colon A{*}B) \vdash M' : C$. Finally, with rule (C-TM) we can derive $\Gamma'(z\colon A{*}B) \vdash f(M') : D[M'/x]$, which is as required.

- (1-TY), (1-I), (1-EQ) are immediate consequences of the induction hypothesis.

- (Π-TY), (Π-I), (Π-E), (Π-TY-CGR), (Π-I-CGR), (Π-E-CGR) (Π-β), (Π-η). These rules follow from the induction hypothesis, making essential use of the commuting conversion rules (BU-CC), (TY-CC) and (TM-CC). We do the case for (Π-E).

$$(\Pi\text{-E})\ \frac{\Phi, u\colon C \vdash D \text{ Type} \qquad \Phi \vdash M : \Pi u\colon C.D \qquad \Phi \vdash N : C}{\Phi \vdash \text{app}_{(u:C)D}(M,N) : D[N/u]}$$

Assume $\Gamma(\Delta) \equiv \Phi$ with $x\colon A * y\colon B \succcurlyeq \Delta$ and $x, y \notin \Gamma_\circ$. The induction hypothesis gives $\Gamma'(z\colon A*B) \vdash M' : E$, $\Gamma''(z\colon A*B) \vdash N' : C'$ and $\Gamma'''(z\colon A*B), u\colon C'' \vdash D''$ Type with $|(\text{let } z \text{ be } x*y \text{ in } \Gamma_\circ)(z\colon A*B)| = |\Gamma'(z\colon A*B)| = |\Gamma''(z\colon A*B)| = |\Gamma'''(z\colon A*B)|$, $|\text{let } z \text{ be } x*y \text{ in } M| = |M'|$, $|\text{let } z \text{ be } x*y \text{ in } N| = |N'|$, $|\text{let } z \text{ be } x*y \text{ in } (\Pi u\colon C.D)| = |E|$, $|\text{let } z \text{ be } x*y \text{ in } C| = |C'| = |C''|$ and $|\text{let } z \text{ be } x*y \text{ in } D| = |D''|$. Using (BU-CC), we obtain $\vdash \Gamma'(z\colon A*B) = \Gamma''(z\colon A*B) = \Gamma'''(z\colon A*B)$ Bunch. By definition of $|-|$, we have $|E| = |\Pi u\colon C''.D''|$. Using (TY-CC), we obtain from this $\Gamma'(z\colon A*B) \vdash E = \Pi u\colon C''.D''$ Type. By conversion, this gives $\Gamma'(z\colon A*B) \vdash M' : \Pi u\colon C''.D''$. Using (TY-CC), we obtain $\Gamma'(z\colon A*B) \vdash C'' = C'$ Type, which by conversion yields $\Gamma'(z\colon A*B) \vdash N' : C''$. Hence, we can use the rule (Π-E) to derive $\Gamma'(z\colon A*B) \vdash \text{app}_{(u:C'')D''}(M', N') : D''[N'/u]$. By definition of $|-|$, this sequent is as required.

- The cases for Σ-types are similar to that for Π-types.
- (Π^*-TY), (Π^*-I), (Π^*-E), (Π^*-E-CGR). We spell out the case for (Π^*-E).

$$(\Pi^*\text{-E})\ \frac{\Phi * u\colon C \vdash D \text{ Type} \qquad \Phi \vdash M : \Pi^* u\colon C.D \qquad \Psi \vdash N : C}{\Phi * \Psi \vdash \text{app}^*_{(u:C)D}(M, N) : D[N/u]}$$

  Assume $\Gamma(\Delta) \equiv (\Phi * \Psi)$ with $x\colon A * y\colon B \succcurlyeq \Delta$ and $x, y \notin \Gamma_\circ$. We distinguish two cases. First, all variables in Δ occur in either Φ or Ψ. We do the case where the variables of Δ occur in Φ. Then there exists $\Gamma'_\circ$ with $\Gamma'(\Delta) \equiv \Phi$ and $x, y \notin \Gamma'_\circ$. We can apply the induction hypothesis to the two left premises to get $\Gamma''(z\colon A*B) * u\colon C' \vdash D'$ Type and $\Gamma'''(z\colon A*B) \vdash M' : E$ with $|(\text{let } z \text{ be } x*y \text{ in } \Gamma'_\circ)(z\colon A*B)| = |\Gamma''(z\colon A*B)| = |\Gamma'''(z\colon A*B)|$, $|C| = |C'|$, $|\text{let } z \text{ be } x*y \text{ in } D| = |D'|$, $|\text{let } z \text{ be } x*y \text{ in } M| = |M'|$ and $|\text{let } z \text{ be } x*y \text{ in } \Pi^* u\colon C.D| = |E|$. By (BU-CC) we have $\vdash \Gamma''(z\colon A*B) = \Gamma'''(z\colon A*B)$ Bunch. We have $|E| = |\Pi^* u\colon C'.D'|$, by definition of $|-|$ and because both C and C' are closed types. Hence, by (TY-CC) and conversion we can derive the sequent $\Gamma''(z\colon A*B) \vdash M' : \Pi^* u\colon C'.D'$. By similar reasoning we obtain $\Psi \vdash N : C'$. We can therefore use (Π^*-E) to derive $\Gamma''(z\colon A*B) * \Psi \vdash \text{app}^*_{(u:C')D'}(M', N) : D'[N/u]$. Up to structural congruence $\equiv$, the context in this sequent is as required, so that we can apply (CONG) to complete the case.

  The second case is where $v(\Phi) \cap v(\Delta) \neq \emptyset$ and $v(\Psi) \cap v(\Delta) \neq \emptyset$ hold. Without loss of generality we may assume that Δ is $(x\colon A) * (y\colon B)$, that x occurs in Φ and that y occurs in Ψ. By the structural congruence $\Gamma(x\colon A * y\colon B) \equiv \Phi * \Psi$ this implies that there exists Γ' satisfying $\Gamma(x\colon A * y\colon B) \equiv \Gamma' * (x\colon A) * (y\colon B)$. Therefore, we can apply the rule ($*$-E$'$) followed by structural rules to obtain a sequent of the required form.

- ($*$-E$'$), ($*$-E$'$-CGR), ($*$-β'), ($*$-η') all follow similarly to (Π^*-E).
- (TY-CC), (TM-CC) follow from the induction hypothesis.

□

Note that the rules (BU-CC), (TY-CC) and (TM-CC) are used essentially in the above proof, for example in the case for (Π-E).

Lemma 7.2.8. *In* $\mathbf{BT}^-$ *the following rule, in which* $\mathscr{K}$ *stands for either* $(A\ \mathrm{Type})$ *or* $(M\colon A)$*, is admissible up to commuting conversion.*

$$\frac{\Delta \vdash M : A \qquad \Phi \vdash N : B \qquad \Gamma(x\colon A * y\colon B) \vdash \mathscr{K}}{\Gamma(\Delta * \Phi) \vdash \mathrm{let}\ M{*}N\ \mathrm{be}\ x{*}y\ \mathrm{in}\ \mathscr{K} = \mathscr{K}[M/x][N/y]}$$

Proof. By induction on derivations, similar to the previous lemma. We sketch the case where the last rule in the derivation of $\Gamma(x\colon A * y\colon B) \vdash \mathscr{K}$ is

$$(\Pi^*\text{-E})\ \frac{\Gamma * x\colon A \vdash B\ \mathrm{Type} \qquad \Phi \vdash M : \Pi^* x\colon C.D \qquad \Psi \vdash N : C}{\Phi * \Psi \vdash \mathrm{app}^*_{(x:C)D}(M,N) : D[N/x]}$$

As in the above proof, we consider two cases. The first case is where both x and y are in Φ or both are in Ψ. In this case, we can apply the induction hypothesis, and the assertion follows by congruence. The remaining case is where one of x and y is in Φ and the other is in Ψ. As above, then we have $\Phi * \Psi \equiv \Gamma' * (x\colon A) * (y\colon B)$. In this case, we can apply rule $(*\text{-}\beta')$. □

Lemma 7.2.9. *In* $\mathbf{BT}^-$ *the following rule, in which* $\mathscr{K}$ *stands for either* $(A\ \mathrm{Type})$ *or* $(M\colon A)$*, is admissible up to commuting conversion.*

$$\frac{\Delta \vdash M : A{*}B \qquad \Gamma(z\colon A{*}B) \vdash \mathscr{K}}{\Gamma(\Delta) \vdash \mathscr{K}[M/z] = \mathrm{let}\ M\ \mathrm{be}\ x{*}y\ \mathrm{in}\ \mathscr{K}[x{*}y/z]}$$

Proof. By induction on derivations, similar to the previous lemma. □

The above lemmas show that the rules of **BT** are all admissible up to commuting conversion in $\mathbf{BT}^-$. Since in both **BT** and $\mathbf{BT}^-$ commuting conversions are provable equalities, by Proposition 7.2.3 and rules (TY-CC) and (TM-CC), this implies that **BT** and $\mathbf{BT}^-$ are essentially equivalent. Since $\mathbf{BT}^-$ does not have the rule (SUBST), translation of derivations in **BT** to $\mathbf{BT}^-$ amounts to elimination of substitution.

Chapter 8

Freefrom Types

Having defined and studied a bunched type theory with Π^*-types, our next goal is to extend it with Σ^*-types. Recall that Π^* and Σ^*-types are essentially given by a fibred adjunction $\Sigma_A^* \dashv W_A \dashv \Pi_A^*$ for each A. When we introduced Π^*-types in Chapter 4, we did not need to give a syntax for W_A, since the adjunction $W_A \dashv \Pi_A^*$ can be captured completely by a one-to-one correspondence between maps in $\mathbb{E}_\Gamma(1, \Pi_A^* B)$ and $\mathbb{E}_{\Gamma * A}(1 \cong W_A 1, B)$, as is explained in Section 2.5.2. For the adjunction $\Sigma_A^* \dashv W_A$, however, such a characterisation is not available. To give a syntax for Σ^*, we need a syntax for W_A.

In this chapter we introduce syntax for working with W_A. The type corresponding to $W_A B$ is written $B^{*(N:A)}$. In the intended model with names, it consists of all the elements of B that are *free from* the names in the value denoted by $N\colon A$. We call the type $B^{*(N:A)}$ a *freefrom* type.

Although the semantics justifies freefrom types $B^{*(N:A)}$ for open types B, at present we only have a good syntax for *closed* freefrom types, by which we mean types $B^{*(N:A)}$ in which both B and A are closed. Since the formulation of Σ^*-types rests on freefrom types, this is a significant restriction on the type theory. In particular, we do not expect that the completeness result from the previous chapter can be generalised to the extensions of the type theory that we consider from now on. In the second half of this chapter we discuss some possibilities for lifting the restrictions on freefrom types.

8.1 The System $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*, B^{*(M:A)})$

8.1.1 Syntax

We extend the syntax of $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$ as follows.

Types	$A ::= \cdots \mid A^{*(M:A)}$
Terms	$::= \cdots \mid M^{*M} \mid \text{let } M \text{ be } x^{*x}\colon A^{*(M:A)} \text{ in } M \mid \text{join}_{A,A,A}(M,M)$

$$FV(B^{*(N:A)}) = FV(B) \cup FV(N) \cup FV(A)$$
$$FV(M^{*N}) = FV(M) \cup FV(N)$$
$$FV(\text{let } M \text{ be } x^{*y}\colon B^{*(N:A)} \text{ in } R) = FV(M) \cup FV(B^{*(N:A)}) \cup (FV(R) \setminus \{x,y\})$$
$$FV(\text{join}_{A,B,C}(M,N)) = FV(A) \cup FV(B) \cup FV(C) \cup FV(M) \cup FV(N)$$

8.1.2 Rules for Closed Freefrom Types

The rules marked with † are discussed in more detail in the next section.

Formation

$$\text{(FF-Ty)}\ \frac{\vdash A \text{ Type} \qquad \vdash B \text{ Type} \qquad \Delta \vdash N : A}{\Delta \vdash B^{*(N:A)} \text{ Type}}$$

Introduction

$$\text{(FF-I)}\ \frac{\vdash A \text{ Type} \qquad \vdash B \text{ Type} \qquad \Gamma \vdash M : B \qquad \Delta \vdash N : A}{\Gamma * \Delta \vdash M^{*N} : B^{*(N:A)}}$$

Elimination

$$\text{(FF-E)}\ \frac{\Delta \vdash M : B^{*(N:A)} \qquad \begin{array}{c}\Gamma(x\colon A, z\colon B^{*(x:A)}) \vdash C \text{ Type} \\ \Gamma(y\colon B * x\colon A)[y^{*x}/z] \vdash R : C[y^{*x}/z]\end{array}}{\Gamma(\Delta)[N/x][M/z] \vdash \text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R : C[N/x][M/z]}$$

Congruences

$$\text{(FF-Ty-Cgr)}\ \frac{\vdash A_1 = A_2 \text{ Type} \qquad \vdash B_1 = B_2 \text{ Type} \qquad \Delta \vdash N_1 = N_2 : A_1}{\Delta \vdash {B_1}^{*(N_1:A_1)} = {B_2}^{*(N_2:A_2)} \text{ Type}}$$

$$\text{(FF-I-Cgr)}\ \frac{\vdash A_1 = A_2 \text{ Type} \qquad \vdash B_1 = B_2 \text{ Type} \qquad \Gamma \vdash M_1 = M_2 : B_1 \qquad \Delta \vdash N_1 = N_2 : A_1}{\Gamma * \Delta \vdash {M_1}^{*N_1} = {M_2}^{*N_2} : {B_1}^{*(N_1:A_1)}}$$

$$\text{(FF-E-Cgr)}\ \frac{\begin{array}{c}\vdash A_1 = A_2 \text{ Type} \\ \vdash B_1 = B_2 \text{ Type} \\ \Delta \vdash N_1 = N_2 : A_1 \qquad \Gamma(x\colon A_1, z\colon {B_1}^{*(x:A_1)}) \vdash C \text{ Type} \\ \Delta \vdash M_1 = M_2 : {B_1}^{*(N_1:A_1)} \qquad \Gamma(y\colon B_1 * x\colon A_1)[y^{*x}/z] \vdash R_1 = R_2 : C[y^{*x}/z]\end{array}}{\Gamma(\Delta)[N_1/x][M_1/z] \vdash \begin{array}{l}\text{let } M_1 \text{ be } y^{*x}\colon {B_1}^{*(N_1:A_1)} \text{ in } R_1 \\ = \text{let } M_2 \text{ be } y^{*x}\colon {B_2}^{*(N_2:A_2)} \text{ in } R_2\end{array} : C[N_1/x][M_1/z]}$$

Equations

$$\text{(FF-}\beta\text{)}\ \frac{\begin{array}{cc}\Gamma \vdash M : B & \Phi(x\colon A, z\colon B^{*(x:A)}) \vdash C\ \text{Type} \\ \Delta \vdash N : A & \Phi(y\colon B * x\colon A)[y^{*x}/z] \vdash R : C[y^{*x}/z]\end{array}}{\Phi(\Gamma * \Delta)[N/x][M^{*N}/z] \vdash \text{let } M^{*N} \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R = R[N/x][M/y] : C[N/x][M^{*N}/z]}$$

$$\text{(FF-}\eta\text{)}\ \frac{\Delta \vdash M : B^{*(N:A)} \qquad \Gamma(x\colon A, z\colon B^{*(x:A)}) \vdash R : C}{\Gamma(\Delta)[N/x][M/z] \vdash R[N/x][M/z] = \text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R[y^{*x}/z] : C[N/x][M/z]}$$

$$\text{(FF-INJECT)}\ \frac{\begin{array}{c}\Gamma \vdash M : B^{*(N:A)} \qquad \Gamma \vdash R : B^{*(N:A)} \\ \Gamma \vdash (\text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } y) = (\text{let } R \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } y) : B\end{array}}{\Gamma \vdash M = R : B^{*(N:A)}}$$

Join rules

$$\text{(FF-JOIN)}^{\dagger}\ \frac{\Gamma \vdash M : B^{*(R:A)} \qquad \Gamma \vdash N : C^{*(R:A)}}{\Gamma \vdash \text{join}_{A,B,C}(M,N) : (B \times C)^{*(R:A)}}$$

$$\text{(FF-JOIN-EQ1)}^{\dagger}\ \frac{\Gamma \vdash M : B^{*(R:A)} \qquad \Gamma \vdash N : C^{*(R:A)}}{\Gamma \vdash (\text{let } \text{join}_{A,B,C}(M,N) \text{ be } y^{*x}\colon (B \times C)^{*(R:A)} \text{ in } \text{fst}(y)^{*x}) = M : B^{*(R:A)}}$$

$$\text{(FF-JOIN-EQ2)}^{\dagger}\ \frac{\Gamma \vdash M : B^{*(R:A)} \qquad \Gamma \vdash N : C^{*(R:A)}}{\Gamma \vdash (\text{let } \text{join}_{A,B,C}(M,N) \text{ be } y^{*x}\colon (B \times C)^{*(R:A)} \text{ in } \text{snd}(y)^{*x}) = N : C^{*(R:A)}}$$

8.1.3 Discussion

We have chosen not to annotate the introduction terms M^{*N} with types, as this makes the presentation more readable. The price to pay for this simplification is a slight loss of generality. For the soundness argument we have to extend the proof of Lemma 6.5.26 with freefrom types. In the case for M^{*N}, we must, in essence, show $\Gamma \vdash_{\mathbf{ES}} M^{*N} = M'^{*N'} : B^{*(N:A)} = B'^{*(N':A')}$ from the induction hypotheses $\Gamma \vdash_{\mathbf{ES}} M = M' : B = B'$ and $\Gamma \vdash_{\mathbf{ES}} N = N' : A = A'$. If all we have is a rule corresponding to (FF-TY-CGR), then we cannot derive the $B^{*(N:A)} = B'^{*(N':A')}$ from the type equations $B = B'$ and $A = A'$ in context Γ. If we annotate the terms M^{*N} with the types B and A then we can get the additional hypotheses $\vdash_{\mathbf{ES}} B = B'$ Type and $\vdash_{\mathbf{ES}} A = A'$ Type, just as in the case for $\text{pair}^*_{A,B}(M,N)$ in Lemma 6.5.26. With these additional assumptions, the required equation follows easily. Without type annotations, we can still get the additional assumptions if the semantics has the additional property that, for all closed types ($\vdash_{\mathbf{ES}} A$ Type) and ($\vdash_{\mathbf{ES}} B$ Type), it holds that ($\Gamma \vdash_{\mathbf{ES}} A = B$ Type) implies ($\vdash_{\mathbf{ES}} A = B$ Type). Since this property holds in the split fibrations constructed in Chapter 3, we prefer to omit the annotations on the introduction terms, thus trading generality for readability. We remark that the same argument can be used for omitting the type annotations on the terms $\text{pair}^*_{A,B}(M,N)$.

The rules for closed freefrom types look quite similar to those for $*$-types. Indeed, the following proposition shows that the rules for $*$-types are derivable from the rules for closed freefrom types. This proposition is a syntactic version of Proposition 2.5.18.

Proposition 8.1.1. *The rules for $*$-types are derivable from the rules for closed freefrom types.*

Proof idea. We can implement $*$-types using freefrom types by means of the following definitions.

$$A*B \stackrel{\text{def}}{=} \Sigma x\colon A.\, B^{*(x:A)}$$
$$\text{pair}^*_{A,B}(M,N) \stackrel{\text{def}}{=} M^{*N}$$
$$\text{let } M \text{ be } x{:}A*y{:}B \text{ in } N \stackrel{\text{def}}{=} \text{let } \text{snd}(M) \text{ be } y^{*x}\colon B^{*(\text{fst}(M):A)} \text{ in } N$$

□

If we assume extensional identity types, then we can also show the converse. When referring to extensional identity types, we assume standard rules such as the ones shown below (plus appropriate congruence rules). See [47] for more information on identity types.

$$(\textsc{Id-Ty})\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash M \simeq_A N\ \text{Type}} \qquad (\textsc{Id-I})\ \frac{\Gamma \vdash M : A}{\Gamma \vdash \text{refl}(M) : M \simeq_A M}$$

$$(\textsc{Id-Defeq})\ \frac{\Gamma \vdash M \simeq_A N\ \text{Type}}{\Gamma \vdash M = N : A} \qquad (\textsc{Id-Uni})\ \frac{\Gamma \vdash p : M \simeq_A N}{\Gamma \vdash p = \text{refl}(M) : M \simeq_A N}$$

Proposition 8.1.2. *The rules for closed freefrom types can be derived from the rules for $*$-types if we assume extensional identity types and the following constants and equations.*

$$x\colon B*A,\, y\colon C*A,\, p\colon \dot\pi_2(x) \simeq_A \dot\pi_2(y) \vdash \text{j}_{A,B,C}(p,x,y) : (B\times C)*A$$
$$x\colon B*A,\, y\colon C*A,\, p\colon \dot\pi_2(x) \simeq_A \dot\pi_2(y) \vdash \dot\pi_1(\text{j}_{A,B,C}(p,x,y)) = \langle \dot\pi_1(x), \dot\pi_1(y)\rangle : B\times C$$
$$x\colon B*A,\, y\colon C*A,\, p\colon \dot\pi_2(x) \simeq_A \dot\pi_2(y) \vdash \dot\pi_2(\text{j}_{A,B,C}(p,x,y)) = \dot\pi_2(x) : A$$

Proof idea. Define freefrom types as follows.

$$B^{*(N:A)} \stackrel{\text{def}}{=} \Sigma p\colon B*A.\, (N \simeq \dot\pi_2(p))$$
$$M^{*N} \stackrel{\text{def}}{=} \langle \text{pair}^*_{B,A}(M,N), \text{refl}(N)\rangle$$
$$\text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R \stackrel{\text{def}}{=} \text{let } \text{fst}(M) \text{ be } y{:}B*x{:}A \text{ in } R$$

In the definition of M^{*N} above, A and B are the unique types ($\vdash A$ Type) and ($\vdash B$ Type) such that $M\colon B$ and $N\colon A$. Such unique types exist because of Corollary 7.2.4 and the assumption that ($\Gamma \vdash A = B$ Type) implies ($\vdash A = B$ Type) for all ($\vdash A$ Type) and ($\vdash B$ Type).

As a representative example, we show that (FF-η) is derivable. With the above translations, the assumptions of this rule are $\Delta \vdash M : (\Sigma p\colon B*A.\, N \simeq \dot\pi_2(p))$ and $\Gamma(x\colon A, z\colon (\Sigma p\colon B*A.\, x \simeq \dot\pi_2(p))) \vdash R : C$. From this we get $\Delta \vdash \text{fst}(M) : B*A$ and $\Gamma(u\colon B*A, z\colon (\Sigma p\colon B*A.\, \dot\pi_2(u) \simeq \dot\pi_2(p)))\,[\dot\pi_2(u)/x] \vdash R\,[\dot\pi_2(u)/x] : C\,[\dot\pi_2(u)/x]$. It follows by substitution that the term $R\,[\dot\pi_2(u)/x]\,[\langle u, \text{refl}(\dot\pi_2(u))\rangle/z]$ has type

$C[\dot{\pi}_2(u)/x][\langle u, \mathrm{refl}(\dot{\pi}_2(u))\rangle/z]$ in context $\Gamma(u\colon B{*}A)[\dot{\pi}_2(u)/x][\langle u, \mathrm{refl}(\dot{\pi}_2(u))\rangle/z]$. We can apply $(*\text{-}\eta)$ to get the equation

$$\begin{aligned} & R[\dot{\pi}_2(\mathrm{fst}(M))/x][\langle \mathrm{fst}(M), \mathrm{refl}(\dot{\pi}_2(\mathrm{fst}(M)))\rangle/z] \\ =& \text{let } \mathrm{fst}(M) \text{ be } y{:}B{*}x{:}A \text{ in } R[\dot{\pi}_2(\mathrm{pair}^*_{B,A}(y,x))/x][\langle \mathrm{pair}^*_{B,A}(y,x), \mathrm{refl}(\dot{\pi}_2(\mathrm{pair}^*_{B,A}(y,x)))\rangle/z]. \end{aligned}$$

From now on we omit the context and type for brevity. By $\dot{\pi}_2(\mathrm{pair}^*_{B,A}(y,x)) = x$, the right-hand side can be simplified to $(\text{let } \mathrm{fst}(M) \text{ be } y{*}x \text{ in } R[\langle \mathrm{pair}^*_{B,A}(y,x), \mathrm{refl}(x)\rangle/z])$. Notice that this term is just the encoding of $(\text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R[y^{*x}/z])$. For the left-hand side, we have $(\dot{\pi}_2(\mathrm{fst}(M)) = N)$ and $\mathrm{refl}(\dot{\pi}_2(\mathrm{fst}(M))) = \mathrm{snd}(M)$, by (ID-DEFEQ) and (ID-UNI). The term on the left-hand side therefore equals $R[N/x][\langle \mathrm{fst}(M), \mathrm{snd}(M)\rangle/z]$, which using $(\Sigma\text{-}\eta)$ is equal to $R[N/x][M/z]$. Hence, we have shown the required equation $R[N/x][M/z] = \text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R[y^{*x}/z]$.

The rules for join are derivable using the constants assumed in the statement of the proposition. □

Extensional equality appears to be necessary in the above proposition. For example, for closed freefrom types we can derive the equality $(\text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } x) = N$ by $(\text{FF-}\eta)$, but from the encoding of freefrom types using $*$-types in the above proof, the most we seem to be able to get is an inhabitant of the propositional equality type $(\text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } x) \simeq N$.

The rules for join deserve comment. Their existence is due to the restriction to closed freefrom types. Because of this restriction, the other rules are not a complete syntax for the functor W_A. One important property of W_A not available without the join rules is that W_A is a fibred functor. We outline what this means by considering the non-split version of W_A in the non-split codomain fibration. Given two closed types $B \to 1$ and $C \to 1$, the square below is a pullback.

$$\begin{array}{ccc} B \times C & \xrightarrow{\pi_2} & C \\ {\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle !} \\ B & \xrightarrow[!]{} & 1 \end{array}$$

The map $(B \times C) \to 1$ corresponds to the closed type $(B \times C)$. By definition, W_A takes this square to the square below. Moreover, since W_A is a fibred functor this square is again a pullback.

$$\begin{array}{ccc} (B \times C) * A & \xrightarrow{\pi_2 * A} & C * A \\ {\scriptstyle \pi_1 * A}\downarrow & \lrcorner & \downarrow{\scriptstyle !*A} \\ B * A & \xrightarrow[!*A]{} & 1 * A \end{array} \tag{8.1}$$

Now, as explained in more detail in Section 8.3 below, the maps $B * A \to 1 * A$, $C * A \to 1 * A$ and $(B \times C) * A \to 1 * A$ in this diagram correspond (up to $A \cong 1 * A$) to the types $(x\colon A \vdash B^{*(x:A)} \text{ Type})$, $(x\colon A \vdash C^{*(x:A)} \text{ Type})$ and $(x\colon A \vdash (B \times C)^{*(x:A)} \text{ Type})$ respectively. That the square is a pullback then amounts to the type-theoretic property that, for any two terms $\Gamma \vdash M : B^{*(R:A)}$ and $\Gamma \vdash N : C^{*(R:A)}$, there exists a unique term $\Gamma \vdash J : (B \times C)^{*(R:A)}$ whose components consist of the parts of M and N. Note

that the superscript in the types of M and N must be the same, since in order to apply the pullback to morphisms $m\colon \Gamma \to B * A$ and $n\colon \Gamma \to C * A$ corresponding to the terms M and N, they must satisfy $(!*A)\circ m = (!*A)\circ n$. In the Schanuel topos, the existence of the term J amounts to the important fact that a pair $\langle x,y\rangle\colon B\times C$ is fresh for some $z\colon A$ if both x is fresh for z and y is fresh for z. The converse follows by functoriality of $*$.

The rules for join make available the information that the square (8.1) is a pullback. The term $\mathrm{join}_{A,B,C}(M,N)$ in (FF-JOIN) corresponds the term J described just above. The reason we have to assume the constants 'join' is the restriction to closed freefrom types. With a formulation of open freefrom types, the rules for join become derivable, see Section 8.4.2.2 below.

8.2 Examples

To give some example derivations with closed freefrom types, we show that rules which use freshness in the style of FreshML 2000 [87] are derivable. These derivations demonstrate why the constant 'join' is needed in practice. They also show the restrictions on closed freefrom types, since the derived rules are less general than those of FreshML 2000.

We consider typing rules in the style of Figure 2 of [87]. We use the following abbreviation.

$$\Gamma, x\colon B^{*(N:A)} \vdash (\downarrow x) \stackrel{\text{def}}{=} \text{let } x \text{ be } u^{*v}\colon B^{*(N:A)} \text{ in } u*v : B*A$$

Abstraction We derive the following rule, similar to the abstraction rule of [87].

$$\frac{\Gamma \vdash c : C \qquad x_1 : A_1, \ldots, x_n : A_n, x\colon A \vdash M : B \qquad \{\Gamma \vdash x_i : {A_i}^{*(c:C)}\}_{1\le i\le n}}{\Gamma \vdash R : (A \to B)^{*(c:C)}}$$

The term R, that we are about to derive, essentially amounts to $\lambda x\colon A.M$.

This rule is less expressive than the corresponding FreshML 2000 rule, because if ${A_i}^{*(c:C)}$ is a type then A_i must be a closed type. To express freshness assumptions as in FreshML 2000, we would, for example, also need to allow A_2 to be of the form ${A_2'}^{*(x_1:A_1)}$.

To derive this rule we start with the following derivation, in which we abbreviate the substitution $\langle \pi_1(z)/x_1\rangle \circ \cdots \circ \langle \pi_n(z)/x_n\rangle$ by σ.

$$\dfrac{\dfrac{\dfrac{\dfrac{x_1 : A_1, \ldots, x_n : A_n, x\colon A \vdash M : B}{x_1 : A_1, \ldots, x_n : A_n \vdash \lambda x\colon A.M : A \to B}}{(x_1 : A_1, \ldots, x_n : A_n) * (c\colon C) \vdash (\lambda x\colon A.M)^{*c} : (A \to B)^{*(c:C)}}}{(z\colon A_1 \times \cdots \times A_n) * (c\colon C) \vdash (\lambda x\colon A.M[\sigma])^{*c} : (A \to B)^{*(c:C)}}}{c\colon C, u\colon (A_1 \times \cdots \times A_n)^{*(c:C)} \vdash \text{let } u \text{ be } z^{*c} \text{ in } (\lambda x\colon A.M[\sigma])^{*c} : (A \to B)^{*(c:C)}}$$

From the assumptions $\Gamma \vdash x_i : {A_i}^{*(c:C)}$ for $1 \le i \le n$ we can derive

$$\Gamma \vdash \mathrm{join}(x_1, \ldots, x_n) : (A_1 \times \cdots \times A_n)^{*(c:C)},$$

where we use a suggestive notation for iterated join. The derivation is completed by weakening with Γ and substituting with c and the join-term.

Application

$$\frac{\Gamma \vdash M : (A \to B)^{*(c:C)} \qquad \Gamma \vdash N : A^{*(c:C)}}{\Gamma \vdash R : B^{*(c:C)}}$$

We derive a term R corresponding to the application of M to N as follows:

$$\dfrac{\dfrac{\Gamma \vdash M : (A \to B)^{*(c:C)} \qquad \Gamma \vdash N : A^{*(c:C)}}{\Gamma \vdash \mathrm{join}(M,N) : ((A \to B) \times A)^{*(c:C)}} \qquad \dfrac{\dfrac{x\colon A \to B, y\colon A \vdash x\,y : B}{(x\colon A \to B, y\colon A) * (c\colon C) \vdash (x\,y)^{*c} : B^{*(c:C)}}}{(p\colon (A \to B) \times A) * (c\colon C) \vdash (\pi_1(p)\ \pi_2(p))^{*c} : B^{*(c:C)}}}{\Gamma \vdash \mathrm{let}\ \mathrm{join}(M,N)\ \mathrm{be}\ p^{*c}\ \mathrm{in}\ (\pi_1(p)\ \pi_2(p))^{*c} : B^{*(c:C)}}$$

The such derived term R is, up to the presence of freefrom-terms, just the application of M to N.

Abstraction for $-\!\!*$-functions

$$\frac{(x_1\colon A_1, \ldots, x_n\colon A_n) * x\colon A \vdash M : B \qquad \{\Gamma \vdash x_i : {A_i}^{*(c:C)}\}_{1 \le i \le n}}{\Gamma \vdash R : (A -\!\!* B)^{*(c:C)}}$$

This rule can be derived by using Π^*-introduction and join as for Π-abstraction above.

Application for $-\!\!*$-functions

$$\frac{\Gamma \vdash M : (A -\!\!* B)^{*(\downarrow N:A*C)} \qquad \Gamma \vdash N : A^{*(c:C)}}{\Gamma \vdash R : B^{*(c:C)}}$$

We start the derivation of the term R as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x\colon A -\!\!* B * y\colon A \vdash x@y : B}{(x\colon A -\!\!* B * y\colon A) * (c\colon C) \vdash (x@y)^{*c} : B^{*(c:C)}}}{(x\colon A -\!\!* B) * (y\colon A * c\colon C) \vdash (x@y)^{*c} : B^{*(c:C)}}}{(x\colon A -\!\!* B) * (z\colon A*C) \vdash \mathrm{let}\ z\ \mathrm{be}\ y*c\ \mathrm{in}\ (x@y)^{*c} : B^{*(\dot\pi_2(z):C)}}}{z\colon A*C, u\colon (A -\!\!* B)^{*(z:A*C)} \vdash \mathrm{let}\ u\ \mathrm{be}\ x^{*z}\ \mathrm{in\ let}\ z\ \mathrm{be}\ y*c\ \mathrm{in}\ (x@y)^{*c} : B^{*(\dot\pi_2(z):C)}}}{\Gamma, u\colon (A -\!\!* B)^{*(\downarrow N:A*C)} \vdash \mathrm{let}\ u\ \mathrm{be}\ x^{*z}\ \mathrm{in\ let}\ z\ \mathrm{be}\ y*c\ \mathrm{in}\ (x@y)^{*c} : B^{*(\dot\pi_2(\downarrow N):C)}}}{\Gamma \vdash \mathrm{let}\ M\ \mathrm{be}\ x^{*z}\ \mathrm{in\ let}\ z\ \mathrm{be}\ y*c\ \mathrm{in}\ (x@y)^{*c} : B^{*(\dot\pi_2(\downarrow N):C)}}$$

To complete the derivation, we show $\Gamma \vdash c = \dot\pi_2(\downarrow N) : C$. To this end we note the following equations of terms of type C in context $c\colon C, z\colon A^{*(c:C)}$.

$$\begin{aligned}
c &= \mathrm{let}\ z\ \mathrm{be}\ y^{*c}\colon A^{*(c:C)}\ \mathrm{in}\ c && \text{by (FF-}\eta\text{)}\\
&= \mathrm{let}\ z\ \mathrm{be}\ y^{*c}\colon A^{*(c:C)}\ \mathrm{in}\ (\mathrm{let}\ y*c\ \mathrm{be}\ u{:}A*v{:}C\ \mathrm{in}\ v) && \text{by } (*\text{-}\beta)\\
&= \mathrm{let}\ z\ \mathrm{be}\ y^{*c}\colon A^{*(c:C)}\ \mathrm{in}\ (\mathrm{let}\ (\mathrm{let}\ y^{*c}\ \mathrm{be}\ y^{*c}\colon A^{*(c:C)}\ \mathrm{in}\ y*c)\ \mathrm{be}\ u{:}A*v{:}C\ \mathrm{in}\ v) && \text{by (FF-}\beta\text{)}\\
&= \mathrm{let}\ (\mathrm{let}\ z\ \mathrm{be}\ y^{*c}\colon A^{*(c:C)}\ \mathrm{in}\ y*c)\ \mathrm{be}\ u{:}A*v{:}C\ \mathrm{in}\ v && \text{by (FF-}\eta\text{)}
\end{aligned}$$

By substitution we obtain $\Gamma \vdash c = \dot\pi_2(\downarrow N) : C$. This gives $B^{*(\dot\pi_2(\downarrow N):C)} = B^{*(c:C)}$, so that we can use type conversion to complete the above derivation.

8.3 Interpretation

In this section we give the interpretation for closed freefrom types. For the particular models constructed in Chapter 3 this is in fact not necessary, since these models are equivalent to codomain fibrations and therefore have extensional equality, so that soundness follows at once from Proposition 8.1.2. We nevertheless spell out the interpretation here, both with an eye to generalisations to open freefrom types and to include the possibility of non-extensional models. Since the rules are incomplete and likely to change, we will be less formal than in Chapter 6.

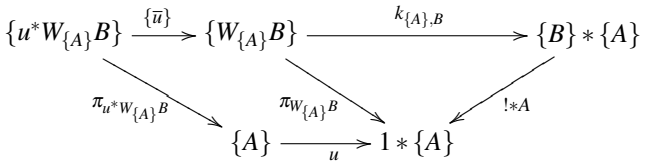
Assume a $(*, 1, \Sigma, \Pi, \Pi^*)$-type-category. We write $p\colon \mathbb{E} \to \mathbb{B}$ for the underlying fibration and denote the associated comprehension functor by $\{-\}\colon \mathbb{E} \to \mathbb{B}$. We write $*$ for the strict affine monoidal structure on $\mathbb{B}$. In addition, we assume a monoidal weakening structure W_A (Definition 2.5.15) on this given structure. For the interpretation of closed freefrom types, we only need to make use of the monoidal weakening functors $W_{\{A\}}\colon \mathbb{E}_1 \to \mathbb{E}_{1*\{A\}}$ for objects A of $\mathbb{E}_1$.

Formation The interpretation of the premises of the type formation rule

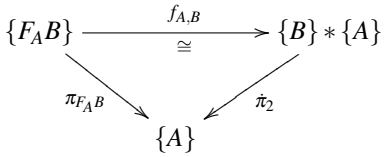
$$\text{(FF-TY)}\ \frac{\vdash A\ \text{Type} \qquad \vdash B\ \text{Type} \qquad \Delta \vdash N : A}{\Delta \vdash B^{*(N:A)}\ \text{Type}}$$

amounts to an object Δ in $\mathbb{B}$, objects A and B in $\mathbb{E}_1$ and a morphism $N\colon \Delta \to \{A\}$ in $\mathbb{B}$. The type $B^{*(N:A)}$ is interpreted as the object $N^* u^* W_{\{A\}} B$ of $\mathbb{E}_\Delta$, where $u\colon \{A\} \to 1 * \{A\}$ is the canonical isomorphism.

By definition of $W_{\{A\}}$ we have an isomorphism $k_{\{A\},B}$ such that the following diagram commutes.



Since u is an isomorphism, so is $\{\overline{u}\}$, and it follows that $f_{A,B} \stackrel{\text{def}}{=} k_{\{A\},B} \circ \{\overline{u}\}$ is also an isomorphism. To simplify the notation, we write F_A for the functor $u^* W_{\{A\}}$. Notice that $F_A B$ is just the interpretation of $\Gamma * x\colon A \vdash B^{*(x:A)}\ \text{Type}$. With the notation $F_A B$, the isomorphism $f_{A,B}$ fits in the following diagram.

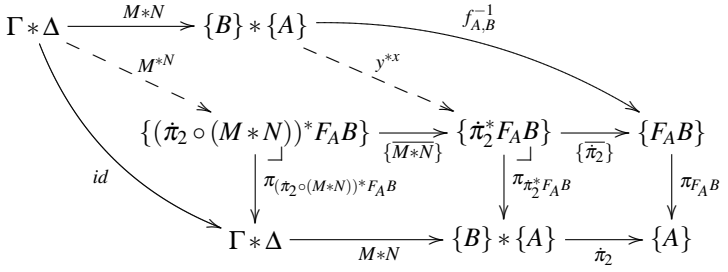


The isomorphism $f_{A,B}$ will be the interpretation of the introduction and elimination terms for freefrom types. Informally, $f^{-1}_{A,B}$ represents the substitution $\langle y^{*x}/z\rangle\colon (y\colon B * x\colon A) \to (x\colon A, z\colon B^{*(x:A)})$ and $f_{A,B}$ represents the let-term $(\text{let } z \text{ be } y^{*x} \text{ in } -)\colon (x\colon A, z\colon B^{*(x:A)}) \to (y\colon B * x\colon A)$.

Introduction The introduction terms are interpreted using $f^{-1}_{A,B}$ and substitution. The assumptions in

$$\text{(FF-I)}\ \frac{\vdash A\ \text{Type} \qquad \vdash B\ \text{Type} \qquad \Gamma \vdash M : B \qquad \Delta \vdash N : A}{\Gamma * \Delta \vdash M^{*N} : B^{*(N:A)}}$$

amount to objects A and B in $\mathbb{E}_1$ as well as maps $M\colon \Gamma \to \{B\}$ and $N\colon \Delta \to \{A\}$ in $\mathbb{B}$. The term in the conclusion of the rule is interpreted as the unique map M^{*N} making the following diagram commute.



(8.2)

Note that $\dot\pi_2 \circ (M * N) = N \circ \dot\pi_2$ holds. This implies that $(\dot\pi_2 \circ (M * N))^* F_A B$ is equal to $(N \circ \dot\pi_2)^* F_A B$, which is just the interpretation of $\Gamma * \Delta \vdash B^{*(N:A)}$ Type.

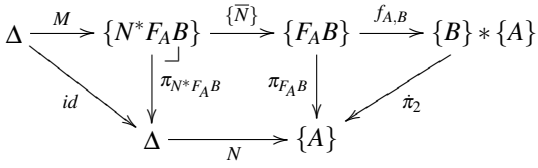
A succinct way of describing the interpretation of M^{*N} is by reindexing $x\colon A, z\colon B^{*(x:A)} \vdash z : B^{*(x:A)}$ along $f^{-1}_{A,B}$, the result of which corresponds to the term $y\colon B * x\colon A \vdash y^{*x} : B^{*(x:A)}$, and then reindexing this term along $M * N$, which amounts to substituting M for y and N for x. The above diagram is nothing but an explication of this description.

Elimination The elimination terms are interpreted using $f_{A,B}$ and substitution. We consider the special case without parameter context:

$$\text{(FF-E)}\ \frac{\Delta \vdash M : B^{*(N:A)} \qquad \begin{array}{c} x\colon A, z\colon B^{*(x:A)} \vdash C \text{ Type} \\ y\colon B * x\colon A \vdash R : C[y^{*x}/z] \end{array}}{\Delta \vdash \text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R : C[N/x][M/z]}$$

The type $(x\colon A, z\colon B^{*(x:A)} \vdash C \text{ Type})$ is interpreted as an object C of $\mathbb{E}_{\{F_A B\}}$. Since the definition of y^{*x} is such that $\{\overline{\dot\pi_2}\} \circ y^{*x} = f^{-1}_{A,B}$ holds (see Diagram 8.2), we know that $(y\colon B * x\colon A \vdash C[y^{*x}/z] \text{ Type})$ is interpreted as $(f^{-1}_{A,B})^* C$ in $\mathbb{E}_{\{B\}*\{A\}}$. The term R is therefore given by a map of type $1_{\{B\}*\{A\}} \to (f^{-1}_{A,B})^* C$ in $\mathbb{E}_{\{B\}*\{A\}}$.

The term M uniquely corresponds to a section, as given by the left triangle in the diagram below. The rest of this diagram commutes by definition.



(8.3)

We interpret the term $(\text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R)$ as the morphism $(f_{A,B} \circ \{\overline{N}\} \circ M)^* R$ of type $1_\Delta \to (f_{A,B} \circ \{\overline{N}\} \circ M)^* (f^{-1}_{A,B})^* C = (f^{-1}_{A,B} \circ f_{A,B} \circ \{\overline{N}\} \circ M)^* C = (\{\overline{N}\} \circ M)^* C$ in $\mathbb{E}_\Delta$. Note that the codomain of this map is the interpretation of the type $C[N/x][M/y]$, as required by the above rule.

In short, the interpretation of $(\text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R)$ is given by reindexing R along $f_{A,B}$ to get $x\colon A, z\colon B^{*(x:A)} \vdash f^*_{A,B} R : C$ and then substituting N for x and M for z.

The general case of (FF-E) with a parameter context $\Gamma_\circ$ follows by lifting the map $f_{A,B} \circ \{\overline{N}\} \circ M$ in the context-with-hole $\Gamma_\circ$ and substituting along the lifted map.

Equations The equations follow because $f_{A,B}$ and $f_{A,B}^{-1}$ are mutually inverse. We start with the β-equation.

$$(\text{FF-}\beta)\ \frac{\begin{array}{ll}\Gamma \vdash M : B & \Phi(x\colon A, z\colon B^{*(x:A)}) \vdash C \text{ Type} \\ \Delta \vdash N : A & \Phi(y\colon B * x\colon A)[y^{*x}/z] \vdash R : C[y^{*x}/z]\end{array}}{\Phi(\Gamma * \Delta)[N/x][M^{*N}/z] \vdash \text{let } M^{*N} \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R = R[N/x][M/y] : C[N/x][M^{*N}/z]}$$

We consider first the case where $\Phi_\circ$ is $\bigcirc$. Since A and B are both closed types (otherwise there would be no valid context containing $B^{*(x:A)}$ as on the right), they correspond to objects of $\mathbb{E}_1$. The terms M and N therefore amount to maps $M\colon \Gamma \to \{B\}$ and $N\colon \Delta \to \{A\}$. The interpretation of $\Gamma * \Delta \vdash M^{*N} : B^{*(N:A)}$ is given by a section $M^{*N}\colon \Gamma * \Delta \to \{(N \circ \dot{\pi}_2)^* F_A B\}$ of $\pi_{(N\circ\dot{\pi}_2)^* F_A B}$. The term $(\text{let } M^{*N} \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R)$ is interpreted as $(f_{A,B} \circ \{\overline{N \circ \dot{\pi}_2}\} \circ M^{*N})^* R$. Now we have:

$$\begin{aligned} f_{A,B} \circ \{\overline{N \circ \dot{\pi}_2}\} \circ M^{*N} &= f_{A,B} \circ \{\overline{\dot{\pi}_2 \circ (M * N)}\} \circ M^{*N} && \text{since } N \circ \dot{\pi}_2 = \dot{\pi}_2 \circ (M * N) \\ &= f_{A,B} \circ \{\overline{\dot{\pi}_2}\} \circ \{\overline{M * N}\} \circ M^{*N} && p \text{ is a split fibration, } \{-\} \text{ a functor} \\ &= f_{A,B} \circ f_{A,B}^{-1} \circ (M * N) && \text{by diagram (8.2)} \\ &= M * N \end{aligned}$$

Therefore, the interpretation of $(\text{let } M^{*N} \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R)$ is $(M * N)^* R$. But since $(M * N)^* R$ is also the interpretation of $R[N/x][M/y]$, this shows the β-equation. The case where $\Phi_\circ$ is not $\bigcirc$ follows by lifting this special case in the context-with-hole $\Phi_\circ$.

We come to the η-equation.

$$(\text{FF-}\eta)\ \frac{\Delta \vdash M : B^{*(N:A)} \qquad \Gamma(x\colon A, z\colon B^{*(x:A)}) \vdash R : C}{\Gamma(\Delta)[N/x][M/z] \vdash R[N/x][M/z] = \text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R[y^{*x}/z] : C[N/x][M/z]}$$

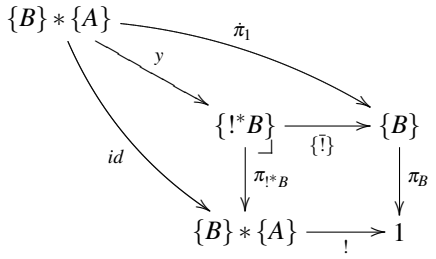
Again, we assume that $\Gamma_\circ$ is $\bigcirc$. In this case, R corresponds to a map $1 \to C$ in $\mathbb{E}_{\{F_A B\}}$. Then, the term $y\colon B * x\colon A \vdash R[y^{*x}/z] : C[y^{*x}/z]$ is interpreted as $(f_{A,B}^{-1})^* R$. The term $(\text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R[y^{*x}/z])$ is therefore interpreted as $(f_{A,B} \circ \{\overline{N}\} \circ M)^* (f_{A,B}^{-1})^* R = (f_{A,B}^{-1} \circ f_{A,B} \circ \{\overline{N}\} \circ M)^* R = (\{\overline{N}\} \circ M)^* R$. But the term $(\{\overline{N}\} \circ M)^* R$ is just the interpretation of $R[N/x][M/z]$, thus showing the η-equation.

It remains to show the equation (FF-INJECT).

$$(\text{FF-INJECT})\ \frac{\begin{array}{l}\Gamma \vdash M : B^{*(N:A)} \qquad \Gamma \vdash R : B^{*(N:A)} \\ \Gamma \vdash (\text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } y) = (\text{let } R \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } y) : B\end{array}}{\Gamma \vdash M = R : B^{*(N:A)}}$$

The interpretations of the terms M and R are uniquely determined by sections $M\colon \Gamma \to \{N^* F_A B\}$ and $R\colon \Gamma \to \{N^* F_A B\}$ of $\pi_{N^* F_A B}$. The equation $(\text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } y) = (\text{let } R \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } y)$ in

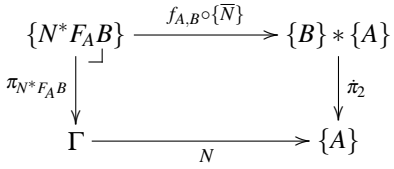
this rule is interpreted as the equation $(f_{A,B} \circ \{\overline{N}\} \circ M)^* y = (f_{A,B} \circ \{\overline{N}\} \circ R)^* y$, in which y is the unique morphism making the following diagram commute.



(8.4)

Note that by definition of reindexing $(f_{A,B} \circ \{\overline{N}\} \circ M)^* y = (f_{A,B} \circ \{\overline{N}\} \circ R)^* y$ implies $y \circ (f_{A,B} \circ \{\overline{N}\} \circ M) = y \circ (f_{A,B} \circ \{\overline{N}\} \circ R)$.

We have to show $M = R$. Since the diagram



(8.5)

is a pullback, and since both M and R are sections of $\pi_{N^* F_A B}$, it suffices to show $f_{A,B} \circ \{\overline{N}\} \circ M = f_{A,B} \circ \{\overline{N}\} \circ R$. Since $*$ is a strict affine monoidal structure, the map $\langle \dot{\pi}_1, \dot{\pi}_2 \rangle \colon \{B\} * \{A\} \to \{B\} \times \{A\}$ is a monomorphism. Therefore, it suffices to show $\dot{\pi}_i \circ f_{A,B} \circ \{\overline{N}\} \circ M = \dot{\pi}_i \circ f_{A,B} \circ \{\overline{N}\} \circ R$ for both $i = 1$ and $i = 2$. For $i = 1$, we use the equation $y \circ (f_{A,B} \circ \{\overline{N}\} \circ M) = y \circ (f_{A,B} \circ \{\overline{N}\} \circ R)$ established above. Since we know from diagram (8.4) that $\dot{\pi}_1 = \{\overline{!}\} \circ y$ holds, post-composing this equation with $\{\overline{!}\}$ yields the required $\dot{\pi}_1 \circ f_{A,B} \circ \{\overline{N}\} \circ M = \dot{\pi}_1 \circ f_{A,B} \circ \{\overline{N}\} \circ R$. For $i = 2$, we calculate as follows:

$$
\begin{aligned}
\dot{\pi}_2 \circ f_{A,B} \circ \{\overline{N}\} \circ M &= N \circ \pi_{N^* F_A B} \circ M && \text{by diagram (8.5)} \\
&= N && \text{since } M \text{ is a section of } \pi_{N^* F_A B} \\
&= N \circ \pi_{N^* F_A B} \circ R && \text{since } R \text{ is a section of } \pi_{N^* F_A B} \\
&= \dot{\pi}_2 \circ f_{A,B} \circ \{\overline{N}\} \circ R && \text{by diagram (8.5)}
\end{aligned}
$$

Hence, we have shown $M = R$, as required.

We omit the details for the interpretation of 'join', which we have already sketched in the previous section.

8.4 Towards Open Freefrom Types

Let us now examine how the syntax for closed freefrom types can be generalised to open types, and let us discuss the difficulties associated with it. We know precisely the semantic structure we want to capture with open freefrom types, namely the fibred functor W_A. For the purpose of this discussion, we consider just the special case of the codomain fibration on the Schanuel topos $\mathbb{S}$. In this special case, the functor $W_A \colon \mathbb{S}^{\to} \to \mathbb{S}^{\to}$ maps $\pi_B \colon B \to \Gamma$ to $\pi_B * A \colon B * A \to \Gamma * A$. Open freefrom types should be a syntax for this functor. The main difficulty in giving such a syntax for W_A is how to account for the substitution behaviour of W_A, i.e. how to integrate in the syntax that W_A is a fibred functor.

To begin with let us consider possible choices for type formation. Since W_A maps $\pi_B \colon B \to \Gamma$ to $\pi_B * A \colon B * A \to \Gamma * A$, a natural choice for an (unsubstituted) type formation rule would be

$$(\text{FFO-TY}) \ \frac{\vdash A \text{ Type} \qquad \Gamma \vdash B \text{ Type}}{\Gamma * x \colon A \vdash B^{*(x:A)} \text{ Type}}$$

The type A must be closed since it corresponds to an object of $\mathbb{S}$. Annotating the type $B^{*(x:A)}$ with the variable x is necessary to keep track of substitutions in x. Some evidence that annotating $B^{*(x:A)}$ with x is also sufficient to give a sound syntax is that, by pullback-preservation of W_A, there is an isomorphism $(\sigma * A)^* W_A \pi_B \cong W_A \sigma^* \pi_B$. This isomorphism states that substitution in the B-part of $B^{*(x:A)}$ can be defined as usual. Without strengthening, however, it does not appear to be quite enough for showing soundness, in particular that different derivations of the same sequent have the same interpretation. Nevertheless, we can rely on the following special property of W_A in (the split fibration equivalent to) the codomain fibration of Schanuel topos to show soundness:

$$\frac{\begin{array}{lll} \sigma \colon \Gamma \to \Delta * x \colon A & \Delta \vdash B \text{ Type} & \Gamma \vdash B[\sigma] = C[\tau] \text{ Type} \\ \tau \colon \Gamma \to \Phi * x \colon A & \Phi \vdash C \text{ Type} & \Gamma \vdash x[\sigma] = x[\tau] : A \end{array}}{\Gamma \vdash (B^{*(x:A)})[\sigma] = (C^{*(x:A)})[\tau] \text{ Type}}$$

This property is similar to the additional property that strong Π^*-types have over normal Π^*-types. It makes it straightforward to show by induction on the size of the type that two derivations have the same interpretation, as in Lemma 6.5.26. In the split fibration for the Schanuel topos, constructed in Chapter 3, this property follows from Proposition 2.5.17. With this special property, we can also show soundness of the following type formation rule

$$(\text{FFO-TY}') \ \frac{\vdash A \text{ Type} \qquad \Gamma \vdash B \text{ Type}}{\Gamma, x \colon A \vdash B^{*(x:A)} \text{ Type}}$$

Semantically, the type in the conclusion amounts to $\iota \circ (\pi_B * A) \colon B * A \to \Gamma * A \rightarrowtail \Gamma \times A$.

It makes a difference whether we choose (FFO-TY) or (FFO-TY$'$). If we choose (FFO-TY) then $*$-elimination on types can no longer be shown to be admissible as in Section 5.4. For example, for a type of the form $z \colon A * B \vdash \text{let } z \text{ be } x * y \text{ in } B(x)^{*(y:A)}$ Type, we cannot push the let further in the subterms. With (FFO-TY$'$), on the other hand, then we can derive $z \colon A * B \vdash B(\text{let } z \text{ be } x * y \text{ in } x)^{*(\text{let } z \text{ be } x * y \text{ in } y:A)}$ Type,

so that we can expect the admissibility argument from Section 5.4 to go through. Another difference between (FFO-TY) and (FFO-TY′) that plays a role in the formulation of the other rules is that the rule (FFO-TY) enforces stronger constraints on the contexts. With (FFO-TY) we can derive $\Gamma \vdash B^{*(N:A)}$ Type only if the variables in B and N are separated in the context Γ. By this we mean that Γ is, up to structural congruence, of the form $\Delta(\Phi * \Psi)$ and both $\Phi \vdash B$ Type and $\Psi \vdash N : A$ are derivable. We will exploit this separation information in the next section below. With the rule (FFO-TY′), however, we do not get any separation information.

To give terms for open freefrom types, it is reasonable to use the isomorphism

$$\begin{array}{ccc} \{W_A B\} & \xrightarrow[\cong]{k_{A,B}} & \{B\} * A \\ & {\scriptstyle \pi_{W_A B}} \searrow \quad \swarrow {\scriptstyle \pi_{B * A}} & \\ & \Gamma * A & \end{array} .$$

This isomorphism is part of the definition of a monoidal weakening structure in Section 2.5.2.2. In the codomain fibration it is in fact the identity. In terms of the syntax, $k_{A,B}$ is an isomorphism between the contexts $(\Gamma * x\colon A), z\colon B^{*(x:A)}$ and $(\Gamma, y\colon B) * x\colon A$ for all types $\vdash A$ Type and $\Gamma \vdash B$ Type. Introduction terms correspond to the inverse of this isomorphism. This inverse may be understood as the substitution

$$\langle y^{*x}/z \rangle\colon (\Gamma, y\colon B) * x\colon A \to (\Gamma * x\colon A), z\colon B^{*(x:A)}.$$

It gives rise to the (unsubstituted) introduction rule

$$\text{(FF-I)}\ \frac{\vdash A \text{ Type} \qquad \Gamma \vdash B \text{ Type}}{(\Gamma, y\colon B) * x\colon A \vdash y^{*x} : B^{*(x:A)}}$$

It is straightforward to close this rule under substitution, and we expect that soundness of this rule can be shown by adding a further rule in the spirit of (INJECT).

The main problem with open freefrom types lies in the formulation of an elimination rule. Given the above isomorphism, we would like to add an (unsubstituted) rule of the following form.

$$\text{(FFO-E)}\ \frac{\ldots \qquad (\Gamma, y\colon B) * x\colon A \vdash R : C[y^{*x}/z]}{(\Gamma * x\colon A), z\colon B^{*(x:A)} \vdash \text{let } z \text{ be } y^{*x}\colon B^{*(x:A)} \text{ in } R : C}$$

Although this rule is very appealing, it is problematic because it is not straightforward to close it under substitution. An instance of this rule that is closed under substitution should have the form

$$\text{(FFO-E-SUBS)}\ \frac{\ldots \qquad \Delta \vdash M : B^{*(N:A)} \qquad (\Gamma, y\colon B) * x\colon A \vdash R : C[y^{*x}/z]}{\Delta \vdash \text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R : C[M/z]}$$

But what should the context Γ be? Clearly it cannot be Δ itself. Suppose Γ were Δ. The above rule would bind N to x, thereby separating Δ from N. But since N is a term in context Δ, we could therefore separate N from itself and this is not possible for all terms N. But perhaps we can somehow obtain Γ from Δ? To some extent, this is possible if we choose (FFO-TY) as the type formation rule; see below for more

detail. In general, however, for example when we choose the rule (FFO-TY′), it is not straightforward to obtain a context Γ from Δ. For example, with (FFO-TY′) we can derive $z\colon A{*}B \vdash C(\dot{\pi}_2(z))^{*(\dot{\pi}_1(z):A)}$ Type for any type $x\colon B \vdash C(x)$ Type, and it is not obvious from the context $z\colon A{*}B$ which context Γ we should choose.

A trivial solution to giving an elimination rule that is closed under substitution is by using explicit substitutions.

$$\dfrac{\begin{array}{lll} & \Delta \vdash M : B^{*(N:A)} & \Delta \vdash B = B'[\sigma] \text{ Type} \\ \ldots & \sigma\colon \Delta \to \Gamma * x\colon A & \Delta \vdash N = x[\sigma] \text{ Type} \\ & \Gamma \vdash B' \text{ Type} & (\Gamma, y\colon B') * x\colon A \vdash R : C[y^{*x}/z] \end{array}}{\Delta \vdash \text{let } M \text{ with } \sigma \text{ be } (\Gamma, y\colon B')^{*x:A} \text{ in } R : C}$$

There are several possible variants of this formulation with explicit substitutions. Instead of making the substitution σ a premise of this rule, the type formation rule for freefrom types could be changed so that σ becomes part of the type $B^{*(N:A)}$. In another direction, it is not necessary to record the full substitution σ in the term of the conclusion. Since $*$ is strict affine, it should be enough just to record the list of terms $x_1[\sigma], \ldots, x_n[\sigma], x[\sigma]$, where $x_1, \ldots, x_n$ are the variables declared in Γ. We also note that σ, as used in the above rule, is similar to the slices of Gabbay & Cheney [37].

At present we do not think that any of the formulations with explicit substitutions leads to a good formulation of freefrom types, since working with the terms and the associated equations is extremely complicated. The examples in [83] illustrate this point for a simply typed modal λ-calculus. Furthermore, the theory of dependent types with named variables and explicit substitutions, as worked out in [109, 32], still appears to have some issues [88]. At present we do not know how to deal with explicit substitutions in the bunched dependent type theory in such a way that the theory remains manageable.

Coming back to the attempt of formulating a rule (FFO-E-SUBS) that is closed under substitution, we note that if we choose (FFO-TY) as the type formation rule for open freefrom types, then it is possible to obtain a context Γ in (FFO-E-SUBS) from Δ. With the rule (FFO-TY), the judgement $\Delta \vdash B^{*(N:A)}$ Type can only be made if Δ has, up to structural congruence, the form $\Phi(\Gamma * \Psi)$ and $\Gamma \vdash B$ Type and $\Psi \vdash N : A$ are derivable. Note, however, that this decomposition of Δ need not be unique. We can use the decomposition of Δ into Γ and Φ in the rule (FFO-E-SUBS) as follows.

$$\dfrac{\ldots \qquad \Phi(\Gamma * \Psi) \vdash M : B^{*(N:A)} \qquad \Gamma \vdash B \text{ Type} \qquad \Psi \vdash N : A \qquad (\Gamma, y\colon B) * x\colon A \vdash R : C[y^{*x}/z]}{\Phi(\Gamma * \Psi) \vdash \text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R : C[M/z]}$$

This elimination rule appears weaker than some of the other possible rules. For example, we cannot use it to eliminate elements of $(\text{let } M \text{ be } x{*}y \text{ in } B^{*(N:A)})$; recall from the above discussion that $*$-elimination on types is not admissible with (FFO-TY). In a formulation with (FFO-TY′), on the other hand, the type $(\text{let } M \text{ be } x{*}y \text{ in } B^{*(N:A)})$ would be provably equal to $(\text{let } M \text{ be } x{*}y \text{ in } B)^{*(\text{let } M \text{ be } x*y \text{ in } N:A)}$. Therefore, a freefrom elimination rule for such a formulation would effectively allow us to eliminate elements of $(\text{let } M \text{ be } x{*}y \text{ in } B^{*(N:A)})$. With (FFO-TY), no such argument appears to be possible. It remains to be seen whether or not the above rule is nevertheless complete, or at least sufficient for practical purposes.

8.4.1 A Formulation of Open Freefrom Types

In this section we give a possible formulation of open freefrom types for the Schanuel topos.

Formation

$$(\text{FFO-TY})\ \frac{\vdash A\ \text{Type} \qquad \Gamma \vdash B\ \text{Type} \qquad \Delta \vdash N : A}{\Gamma * \Delta \vdash B^{*(N:A)}\ \text{Type}}$$

Introduction

$$(\text{FFO-I})\ \frac{\vdash A\ \text{Type} \qquad \Gamma \vdash M : B \qquad \Delta \vdash N : A}{\Gamma * \Delta \vdash M^{*N} : B^{*(N:A)}}$$

Elimination

$$(\text{FFO-E})\ \frac{\begin{array}{ccc}\Gamma \vdash B\ \text{Type} & & (\Gamma * x\colon A), z\colon B^{*(x:A)} \vdash C\ \text{Type} \\ \Delta \vdash N : A & \Phi(\Gamma * \Delta) \vdash M : B^{*(N:A)} & (\Gamma, y\colon B) * x\colon A \vdash R : C[y^{*x}/z]\end{array}}{\Phi(\Gamma * \Delta) \vdash \text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R : C[N/x][M/z]}$$

Congruences

$$(\text{FFO-TY-CGR})\ \frac{\vdash A_1 = A_2\ \text{Type} \qquad \Gamma \vdash N_1 = N_2 : A_1 \qquad \Gamma \vdash B_1 = B_2\ \text{Type}}{\Gamma \vdash {B_1}^{*(N_1:A_1)} = {B_2}^{*(N_2:A_2)}\ \text{Type}}$$

$$(\text{FFO-I-CGR})\ \frac{\vdash A\ \text{Type} \qquad \Gamma \vdash M_1 = M_2 : B \qquad \Delta \vdash N_1 = N_2 : A}{\Gamma * \Delta \vdash {M_1}^{*N_1} = {M_2}^{*N_2} : B^{*(N_1:A)}}$$

$$(\text{FFO-E-CGR})\ \frac{\begin{array}{ccc}\vdash A_1 = A_2\ \text{Type} & & \Delta \vdash N_1 = N_2 : A_1 \\ \Gamma \vdash B_1 = B_2\ \text{Type} & & \Phi(\Gamma * \Delta) \vdash M_1 = M_2 : {B_1}^{*(N_1:A_1)} \\ (\Gamma * x\colon A_1), z\colon {B_1}^{*(x:A_1)} \vdash C\ \text{Type} & (\Gamma, y\colon B_1) * x\colon A_1 \vdash R_1 = R_2 : C[y^{*x}/z]\end{array}}{\Phi(\Gamma * \Delta) \vdash \text{let } M_1 \text{ be } y^{*x}\colon {B_1}^{*(N_1:A_1)} \text{ in } R_1 = \text{let } M_2 \text{ be } y^{*x}\colon {B_2}^{*(N_2:A_2)} \text{ in } R_2 : C[N/x][M/z]}$$

Equations

$$(\text{FFO-}\beta)\ \frac{\begin{array}{ccc}\Gamma \vdash M : B & \Delta \vdash N : A & (\Gamma * x\colon A), z\colon B^{*(x:A)} \vdash C\ \text{Type} \\ & & (\Gamma, y\colon B) * x\colon A \vdash R : C[y^{*x}/z]\end{array}}{\Gamma * \Delta \vdash \text{let } M^{*N} \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } R = R[N/x][M/y] : C[M^{*N}/z]}$$

$$(\text{FFO-}\eta)\ \frac{\Gamma \vdash B\ \text{Type} \qquad \Delta \vdash N : A \qquad \Phi(\Gamma * \Delta) \vdash M : B^{*(N:A)} \qquad (\Gamma * x\colon A), z\colon B^{*(x:A)} \vdash R : C}{\Phi(\Gamma * \Delta) \vdash R[N/x][M/z] = \text{let } M \text{ be } y^{*x}\colon B^{*(N:A)} \text{ in } (R[y^{*x}/z]) : C[N/x][M/z]}$$

$$\text{(FFO-INJECT)}\ \frac{\begin{array}{ll}\Gamma \vdash B\ \text{Type} & \Phi(\Gamma * \Delta) \vdash M : B^{*(N:A)} \\ \Delta \vdash N : A & \Phi(\Gamma * \Delta) \vdash R : B^{*(N:A)} \\ \Phi(\Gamma * \Delta) \vdash (\text{let } M \text{ be } y^{*x} \colon B^{*(N:A)} \text{ in } y) = (\text{let } R \text{ be } y^{*x} \colon B^{*(N:A)} \text{ in } y) : B & \end{array}}{\Phi(\Gamma * \Delta) \vdash M = R : B^{*(N:A)}}$$

As we have noted in the previous section, the formulation of the rule (FF-TY) is such that $*$-elimination on types is no longer admissible. Therefore, we now make the admissible rules ($*$-E-TY), ($*$-E-TY-CGR), ($*$-β-TY), ($*$-η-TY), ($*$-E$^+$) from Chapter 5 primitive rules. Note that the soundness argument for $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*)$ from Chapter 6 goes through without complication with these rules made primitive. In fact, it would also be sensible to formulate a version of (FFO-E) on types, but for the present purposes this is not necessary.

8.4.1.1 Discussion

For brevity, the elimination rules are formulated without parameter contexts. Using $*$, Π and Π^*-types, it can be shown that rules with parameter contexts are admissible; see Sections 4.3 and 5.6 for more information.

The main weak point of the above formulation of open freefrom types is the elimination rule. The problem is that in (FFO-E) there may, up to structural congruence, be many choices of the contexts Γ and Δ. The choice of contexts is not reflected in the term. This makes the term $(\text{let } M \text{ be } y^{*x} \colon B^{*(N:A)} \text{ in } R)$ hard to understand, since it does not tell us which variables x is fresh for. Furthermore, given a derivable term it is not immediately clear how to construct a derivation for it. In Section 8.4.2.2 below we give an example illustrating this problem of having many choices in the rule (FFO-E).

Open freefrom types can help to simplify the rules for Π^*-types. When we introduced $*$-types in Chapter 5, we had to strengthen the congruence rule for Π^*-types to (Π^*-E-CGR$^+$), in order to be able to show completeness of the theory. With open freefrom types it should not be necessary to strengthen this rule. In essence, (Π^*-E-CGR$^+$) allows us to conclude $\mathrm{app}^*_{(x:A)B}(M,x)[\sigma] = \mathrm{app}^*_{(x:A)B'}(N,x)[\tau]$ from $M[\sigma] = N[\tau]$ and $x[\sigma] = x[\tau]$. We sketch how it should be possible to do this with open freefrom types:

$$\begin{array}{ll}
\mathrm{app}^*_{(x:A)B}(M,x)[\sigma] & \\
= (\text{let } M^{*x} \text{ be } u^{*v} \colon (\Pi^* x \colon A.B)^{*(x:A)} \text{ in } u@v)[\sigma] & \text{by (FFO-}\beta\text{)} \\
= \text{let } (M^{*x})[\sigma] \text{ be } u^{*v} \colon (\Pi^* x \colon A.B)[\sigma]^{*(x[\sigma]:A)} \text{ in } u@v & \text{by commuting conversion} \\
= \text{let } (N^{*x})[\tau] \text{ be } u^{*v} \colon (\Pi^* x \colon A.B)[\sigma]^{*(x[\sigma]:A)} \text{ in } u@v & \text{see below} \\
= \text{let } (N^{*x})[\tau] \text{ be } u^{*v} \colon (\Pi^* x \colon A.B')[\tau]^{*(x[\tau]:A)} \text{ in } u@v & \text{premises of } (\Pi^*\text{-E-CGR}^+), \text{ injectivity of } \Pi^* \\
= (\text{let } N^{*x} \text{ be } u^{*v} \colon (\Pi^* x \colon A.B')^{*(x:A)} \text{ in } u@v)[\tau] & \text{by commuting conversion} \\
= \mathrm{app}^*_{(x:A)B'}(N,x)[\tau] & \text{by (FFO-}\beta\text{)}
\end{array}$$

Before we show the third step in this chain of equations, we point out that a formulation of open freefrom types with (FFO-TY) does not appear to be quite sufficient for these equations. In the above equations,

there appears the type $(\Pi^* x\colon A.\, B)[\sigma]^{*(x[\sigma]:A)}$, which to form we need (FFO-TY′). The third step follows because we can show $(M^{*x})[\sigma] = (N^{*x})[\tau]$ using the rule (FFO-INJECT). By this rule, it suffices to show $(\text{let } (M^{*x})[\sigma] \text{ be } u^{*v} \text{ in } u) = (\text{let } (N^{*x})[\tau] \text{ be } u^{*v} \text{ in } u)$. Note that, because of the assumption $x[\sigma] = x[\tau]$, the types of $(M^{*x})[\sigma]$ and $(N^{*x})[\tau]$ have the same superscript, as required by (FFO-INJECT). The required equation can be shown as follows.

$$\begin{aligned}
\text{let } (M^{*x})[\sigma] \text{ be } u^{*v} \text{ in } u &= (\text{let } M^{*x} \text{ be } u^{*v} \text{ in } u)[\sigma] && \text{by CC}\\
&= M[\sigma] && \text{by (FFO-}\beta\text{)}\\
&= N[\tau] && \text{by assumption}\\
&= \text{let } (N^{*x})[\tau] \text{ be } u^{*v} \text{ in } u && \text{as for } (M^{*x})[\sigma]
\end{aligned}$$

We do not spell out the interpretation of the above rules for open freefrom types in the Schanuel topos, but remark that the problem that (FFO-E) admits many different derivations of the same sequent can be dealt with by using Lemma 2.5.13. The rest of the interpretation is a straightforward generalisation of that for closed freefrom types.

8.4.2 Examples for Open Freefrom Types

We give some examples of what can be done with the rules for open freefrom types from Section 8.4.1.

8.4.2.1 Nested freefrom types

We derive an isomorphism $C^{*(z:A*B)} \cong \text{let } z \text{ be } x*y \text{ in } \left(C^{*(x:A)}\right)^{*(y:B)}$ for closed types A, B and C. From left to right, we derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{u\colon C \vdash u : C \qquad x\colon A \vdash x : A}{u\colon C * x\colon A \vdash u^{*x} : C^{*(x:A)}}}{u\colon C * x\colon A * y\colon B \vdash \left(u^{*x}\right)^{*y} : \left(C^{*(x:A)}\right)^{*(y:B)}}}{u\colon C * z\colon A*B \vdash \text{let } z \text{ be } x*y \text{ in } \left(u^{*x}\right)^{*y} : \text{let } z \text{ be } x*y \text{ in } \left(C^{*(x:A)}\right)^{*(y:B)}}}{z\colon A*B,\, v\colon C^{*(z:A*B)} \vdash \text{let } v \text{ be } u^{*z}\colon C^{*(z:A*B)} \text{ in } \left(\text{let } z \text{ be } x*y \text{ in } \left(u^{*x}\right)^{*y}\right) : \text{let } z \text{ be } x*y \text{ in } \left(C^{*(x:A)}\right)^{*(y:B)}}$$

In the other direction, the derivation goes as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{u\colon C \vdash u : C \qquad x\colon A * y\colon B \vdash x*y : A*B}{u\colon C * x\colon A * y\colon B \vdash u^{*x*y} : C^{*(x*y:A*B)}}}{(x\colon A, v\colon C^{*(x:A)}) * y\colon B \vdash M_1 : C^{*(x*y:A*B)}}}{(x\colon A * y\colon B), w\colon \left(C^{*(x:A)}\right)^{*(y:B)} \vdash M_2 : C^{*(x*y:A*B)}}}{z\colon A*B, w\colon \text{let } z \text{ be } x*y \text{ in } \left(C^{*(x:A)}\right)^{*(y:B)} \vdash M_3 : \text{let } z \text{ be } x*y \text{ in } C^{*(x*y:A*B)}}}{z\colon A*B, w\colon \text{let } z \text{ be } x*y \text{ in } \left(C^{*(x:A)}\right)^{*(y:B)} \vdash M_3 : C^{*(z:A*B)}}$$

The terms in this derivation are:

$$M_1 = \text{let } v \text{ be } u^{*x} \colon C^{*(x:A)} \text{ in } u^{*x*y}$$
$$M_2 = \text{let } w \text{ be } v^{*y} \colon \left(C^{*(x:A)}\right)^{*(y:B)} \text{ in } M_1$$
$$M_3 = \text{let } z \text{ be } x*y \text{ in } M_2$$

The last step in the derivation follows by type conversion, since, for a closed type C, we have the equation $(\text{let } z \text{ be } x*y \text{ in } C^{*(x*y:A*B)}) = C^{*(z:A*B)}$.

8.4.2.2 Deriving the rules for join

For closed types A, B and C, we derive a term

$$x\colon A, y\colon B^{*(x:A)}, z\colon C^{*(x:A)} \vdash J : (B \times C)^{*(x:A)}$$

that satisfies the equations for join. The derivation goes as follows.

$$\dfrac{\dfrac{\dfrac{u\colon B, v\colon C \vdash \langle u,v\rangle : B\times C \qquad x_2\colon A \vdash x_2 : A}{(u\colon B, v\colon C) * x_2 \colon A \vdash \langle u,v\rangle^{*x_2} : (B\times C)^{*(x_2:A)}}}{(u\colon B * x_1 \colon A), z\colon C^{*(x_1:A)} \vdash \text{let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } \langle u,v\rangle^{*x_2} : (B\times C)^{*(x_1:A)}}}{x\colon A, y\colon B^{*(x:A)}, z\colon C^{*(x:A)} \vdash \text{let } y \text{ be } u^{*x_1} : B^{*(x:A)} \text{ in let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } \langle u,v\rangle^{*x_2} : (B\times C)^{*(x:A)}}$$

In this derivation we have made implicit use of the derivable equation $(\text{let } R \text{ be } y^{*x} \colon E^{*(Q:F)} \text{ in } x) = Q$. For example, to use the rule (FFO-E) in the second step of the derivation, we must bring the type $(B\times C)^{*(x_2:A)}$ in the form $D[v^{*x_2}/z]$ for some type D. We can take D to be $(B\times C)^{*(\text{let } z \text{ be } a^{*b} \colon C^{*(x_2:A)} \text{ in } b:A)}$. With the equation $(\text{let } R \text{ be } y^{*x} \colon E^{*(Q:F)} \text{ in } x) = Q$, it follows that $D[v^{*x_2}/z]$ is derivably equal to $(B\times C)^{*(x_2:A)}$, as required for the conclusion of the second step in the derivation.

The above derivation is an example in which we have to guess the contexts Γ and Δ in (FFO-E). The second step in this derivation is an instance of the rule (FFO-E) in which Γ and Δ are chosen appropriately. A bad choice of Γ and Δ at this point could give, for example, the following instance of (FFO-E).

$$\dfrac{v\colon C * x_2 \colon A \vdash \langle u,v\rangle^{*x_2} : (B\times C)^{*(x_2:A)}}{(u\colon B * x_1 \colon A), z\colon C^{*(x_1:A)} \vdash \text{let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } \langle u,v\rangle^{*x_2} : (B\times C)^{*(x_1:A)}}$$

The premise of this rule instance is clearly not derivable.

We check that the equation (FF-JOIN-EQ1) is derivable with this definition. Since we have the substitution rule (SUBST), it suffices to derive the equation

$$\text{let } (\text{ let } y \text{ be } u^{*x_1} : B^{*(x:A)} \text{ in let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } \langle u,v\rangle^{*x_2} \text{ }) \text{ be } p^{*q} \colon (B\times C)^{*(x:A)} \text{ in } \mathrm{fst}(p)^{*q} = y \tag{8.6}$$

in context $x\colon A, y\colon B^{*(x:A)}, z\colon C^{*(x:A)}$.

We start by showing the following commuting conversion

$$\begin{aligned}&\text{let } (\text{let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } \langle u, v\rangle^{*x_2}) \text{ be } p^{*q} : (B\times C)^{*(x_1:A)} \text{ in } \text{fst}(p)^{*q}\\ &= \text{let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } (\text{let } \langle u, v\rangle^{*x_2} \text{ be } p^{*q} : (B\times C)^{*(x_2:A)} \text{ in } \text{fst}(p)^{*q})\end{aligned} \tag{8.7}$$

in context $(u : B * x_1 : A), z : C^{*(x_1:A)}$. This equation can be derived in the following two steps.

$$\begin{aligned}&\text{let } (\text{let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } \langle u, v\rangle^{*x_2}) \text{ be } p^{*q} : (B\times C)^{*(x_1:A)} \text{ in } \text{fst}(p)^{*q}\\ &= \text{let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } (\text{let } (\text{let } v^{*x_2} \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } \langle u, v\rangle^{*x_2}) \text{ be } p^{*q} : (B\times C)^{*(x_2:A)} \text{ in } \text{fst}(p)^{*q})\\ &\qquad \text{by (FFO-}\eta\text{)}\\ &= \text{let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } (\text{let } \langle u, v\rangle^{*x_2} \text{ be } p^{*q} : (B\times C)^{*(x_2:A)} \text{ in } \text{fst}(p)^{*q})\\ &\qquad \text{by (FFO-}\beta\text{) and (FFO-E-CGR)}\end{aligned}$$

Using commuting conversions such as (8.7), we can derive the required equation (8.6) as follows.

$$\begin{aligned}&\text{let } (\text{let } y \text{ be } u^{*x_1} : B^{*(x:A)} \text{ in let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } \langle u, v\rangle^{*x_2}) \text{ be } p^{*q} : (B\times C)^{*(x:A)} \text{ in } \text{fst}(p)^{*q}\\ &= \text{let } y \text{ be } u^{*x_1} : B^{*(x:A)} \text{ in let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in let } \langle u, v\rangle^{*x_2} \text{ be } p^{*q} : (B\times C)^{*(x:A)} \text{ in } \text{fst}(p)^{*q}\\ &\qquad \text{by commuting conversion such as (8.7)}\\ &= \text{let } y \text{ be } u^{*x_1} : B^{*(x:A)} \text{ in let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } \text{fst}(\langle u, v\rangle)^{*x_2}\\ &\qquad \text{by (FFO-}\beta\text{) and (FFO-E-CGR)}\\ &= \text{let } y \text{ be } u^{*x_1} : B^{*(x:A)} \text{ in let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } u^{*x_2}\\ &\qquad \text{by } (\Sigma\text{-}\beta 1)\text{, (FFO-I-CGR) and (FFO-E-CGR)}\\ &= \text{let } y \text{ be } u^{*x_1} : B^{*(x:A)} \text{ in } u^{*x_1}\\ &\qquad \text{see below}\\ &= y\\ &\qquad \text{by (FFO-}\eta\text{)}\end{aligned}$$

The step before the last one can be seen from the following derivation.

$$\text{(FFO-}\eta\text{)}\ \frac{\dfrac{\dfrac{u : B \vdash u : B \qquad x_2 : A \vdash x_2 : A}{u : B * x_2 : A \vdash u^{*x_2} : B^{*(x_2:A)}}}{(u : B * x_2 : A), z : C^{*(x_2:A)} \vdash u^{*x_2} : B^{*(x_2:A)}}}{(u : B * x_1 : A), z : C^{*(x_1:A)} \vdash u^{*x_1} = \text{let } z \text{ be } v^{*x_2} : C^{*(x_1:A)} \text{ in } u^{*x_2} : B^{*(x_1:A)}}$$

We believe this example illustrates well the point that it can be hard to understand the let-terms for the above formulation open freefrom types, especially when looked at without a typing derivation.

8.5 Further and Related Work

The most obvious point for further work is the restriction to closed freefrom types. That this is a real restriction can be seen on the examples in this chapter. For example, FreshML 2000 is simply-typed, so that one might expect to get away with closed freefrom types. However, without nested freefrom types $\left(A^{*(n:\mathbf{N})}\right)^{*(m:\mathbf{N})}$ we cannot derive the full rules. We could address this by using $A^{*(n*m:\mathbf{N}*\mathbf{N})}$ instead of the nested freefrom type, but for this to work we then also have to extend the rules for closed freefrom types along the lines of the rule below. Such a reformulation appears to be rather awkward.

$$\text{(FF-I)}\ \frac{\vdash C\ \text{Type} \qquad \Gamma \vdash M : B^{*(N:A)} \qquad \Delta \vdash R : C}{\Gamma * \Delta \vdash M^{*N} : B^{*(N*R:A*C)}}$$

Another example of a situation where closed freefrom types are not enough is Atkey's λ-calculus for resource separation [5]. This calculus uses contexts with specific freshness assertions. Although closed freefrom types can be used to represent such contexts, they are most likely not enough to make the rules of [5] admissible.

Giving a λ-calculus for resource separation, such as those in [87, 5, 80], is not an easy task. Perhaps the fact that a good formulation of open freefrom types type systems would subsume such calculi means that finding such a good formulation really is a hard problem.

On the other hand, the categorical formulation of freefrom types could hardly be simpler. In the codomain fibration on the Schanuel topos it is just the functor mapping $f\colon B \to \Gamma$ to $f*A\colon B*A \to \Gamma*A$.

It is possible that the problem with open freefrom types appears just in our particular formulation of the type theory with bunches. Perhaps a different kind of context structure should be investigated, possibly in the spirit of modal and linear type systems [10, 83, 7]. For example, Pfenning and Wong [83] give a simple term calculus for the modal logic S_4 that works without explicit substitutions by making use of special context stacks. In contrast, earlier systems, such as [11], were quite complicated because they contain explicit substitutions.

Another possibility is that the formulation of freefrom types with explicit substitutions is not as unmanageable as it appears. The main difficulty of the formulation of freefrom types with explicit substitutions is that working with the equational theory becomes very complicated. However, we conjecture that in the Schanuel topos it is not even necessary to look at the term constructors for freefrom types when checking equality. It appears reasonable that for equality checking all terms for freefrom types can be removed and the equality be checked on the resulting simple terms. But working this out remains for future work.

Chapter 9

Simple Monoidal Sums

Having introduced freefrom types as a (somewhat restricted) syntax for the functor W_A, we can now formulate rules for simple monoidal sums Σ^*_A.

In this chapter we introduce Σ^*-types based on our formulation of closed freefrom types. We recall from the introduction that Σ^*-types are non-standard sum types, just as Π^*-types are non-standard product types. In the Schanuel topos, the elements of $\Sigma^* x\colon A.B$ are pairs $M.N$ of elements $M\colon A$ and $N\colon B[M/x]$ such that the names in M are hidden in the pair $M.N$. This name-hiding may also be understood as name-binding, in the sense that the identity of the bound name in an α-equivalence class is hidden. Indeed, the elements of type $\Sigma^* n\colon \mathbf{N}.B$, where $\mathbf{N}$ is the type of names, represent α-equivalence classes of elements of B with respect to a single name. We will study the special case of Σ^*-types of the form $\Sigma^* n\colon \mathbf{N}.B$ in the shape of H-types the next chapters. Therefore, we do not give examples of the use of Σ^*-types in this chapter, referring instead to the examples for H-types in the following chapters.

9.1 The System $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*,B^{*(M:A)},\Sigma^*)$

9.1.1 Syntax

We extend the syntax of $\mathbf{BT}(*,1,\Sigma,\Pi,\Pi^*,B^{*(M:A)})$ with the following types and terms.

$$\begin{array}{ll} \text{Types} & A ::= \cdots \mid \Sigma^* x\colon A.A \\ \text{Terms} & \quad ::= \cdots \mid \mathrm{bind}_{(x:A)A}(M,M) \mid \text{let } M \text{ be } x.x\colon \Sigma^* x\colon A.A \text{ in } M \end{array}$$

$$\begin{aligned} FV(\Sigma^* x\colon A.B) &= FV(A) \cup (FV(B) \setminus \{x\}) \\ FV(\mathrm{bind}_{(x:A)B}(M,N)) &= FV(A) \cup (FV(B) \setminus \{x\}) \cup FV(M) \cup FV(N) \\ FV(\text{let } M \text{ be } x.y\colon \Sigma^* x\colon A.B \text{ in } N) &= FV(M) \cup FV(\Sigma^* x\colon A.B) \cup (FV(N) \setminus \{x,y\}) \end{aligned}$$

9.1.2 Rules for Simple Monoidal Sums

The rules marked with † are discussed in more detail in the next section.

Formation

$$(\Sigma^*\text{-TY})\ \frac{\Gamma * x\colon A \vdash B\ \text{Type}}{\Gamma \vdash \Sigma^* x\colon A.B\ \text{Type}}$$

Introduction

$$(\Sigma^*\text{-I})^\dagger\ \frac{x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : B[M/x]}{\Gamma \vdash \text{bind}_{(x:A)B}(M,N) : (\Sigma^* x\colon A.B)^{*(M:A)}}$$

Abbreviation: $M.N \stackrel{\text{def}}{=} \text{let bind}_{(x:A)B}(M,N)\ \text{be}\ y^{*x}\ \text{in}\ y.$

Elimination

$$(\Sigma^*\text{-E})^\dagger\ \frac{\Gamma \vdash M : \Sigma^* x\colon A.B \qquad (\Gamma * x\colon A), y\colon B \vdash N : C^{*(x:A)}}{\Gamma \vdash \text{let}\ M\ \text{be}\ x.y\colon \Sigma^* x\colon A.B\ \text{in}\ N : C}$$

Congruences

$$(\Sigma^*\text{-TY-CGR})\ \frac{\vdash A_1 = A_2\ \text{Type} \qquad \Gamma * x\colon A_1 \vdash B_1 = B_2\ \text{Type}}{\Gamma \vdash \Sigma^* x\colon A_1.B_1 = \Sigma^* x\colon A_2.B_2\ \text{Type}}$$

$$(\Sigma^*\text{-I-CGR})^\dagger\ \frac{\begin{array}{rr}\vdash A_1 = A_2\ \text{Type} & \Gamma \vdash M_1 = M_2 : A_1 \\ x\colon A_1 \vdash B_1 = B_2\ \text{Type} & \Gamma \vdash N_1 = N_2 : B_1[M_1/x]\end{array}}{\Gamma \vdash \text{bind}_{(x:A_1)B_1}(M_1,N_1) = \text{bind}_{(x:A_2)B_2}(M_2,N_2) : (\Sigma^* x\colon A_1.B_1)^{*(M_1:A_1)}}$$

$$(\Sigma^*\text{-E-CGR})\ \frac{\begin{array}{rr}\vdash A_1 = A_2\ \text{Type} & \Gamma \vdash M_1 = M_2 : \Sigma^* x\colon A_1.B_1 \\ \Gamma * x\colon A_1 \vdash B_1 = B_2\ \text{Type} & (\Gamma * x\colon A_1), y\colon B_1 \vdash N_1 = N_2 : C^{*(x:A_1)}\end{array}}{\Gamma \vdash \text{let}\ M_1\ \text{be}\ x.y\colon \Sigma^* x\colon A_1.B_1\ \text{in}\ N_1 = \text{let}\ M_2\ \text{be}\ x.y\colon \Sigma^* x\colon A_2.B_2\ \text{in}\ N_2 : C}$$

Equations

$$(\Sigma^*\text{-}\beta)^\dagger\ \frac{x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : B[M/x] \qquad x\colon A, y\colon B \vdash R : C^{*(x:A)}}{\Gamma \vdash \text{let bind}_{(x:A)B}(M,N)\ \text{be}\ u^{*v}\colon (\Sigma^* x\colon A.B)^{*(M:A)}\ \text{in}\ (\text{let}\ u\ \text{be}\ x.y\ \text{in}\ R)^{*v} = R[M/x][N/y] : C^{*(M:A)}}$$

$$(\Sigma^*\text{-}\eta)^\dagger\ \frac{\vdash C\ \text{Type} \qquad x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : \Sigma^* x\colon A.B \qquad z\colon \Sigma^* x\colon A.B \vdash N : C}{\Gamma \vdash N[M/z] = \text{let}\ M\ \text{be}\ x.y\ \text{in}\ (\text{let bind}(x,y)\ \text{be}\ z^{*x}\colon (\Sigma^* x\colon A.B)^{*(x:A)}\ \text{in}\ N^{*x}) : C}$$

9.2 Discussion

The rules for Σ^*-types are similar to the standard rules for weak Σ-types, as in e.g. [56, §10.1]. The only real difference is that freefrom types appear in the rules for Σ^*. The use of freefrom types can be explained with the informal view of the elements of Σ^*-types as pairs with hidden names. The introduction rule (Σ^*-I) lets us form an element of $\Sigma^* x\colon A.B$ from a pair of terms $M\colon A$ and $N\colon B[M/x]$. The so formed element of $\Sigma^* x\colon A.B$, however, is not just the pair $\langle M,N\rangle$, but it is the pair $\langle M,N\rangle$ with all the names of M hidden. This name-hiding information is made available by the freefrom type in the introduction rule (Σ^*-I). The term $\mathrm{bind}_{(x:A)B}(M,N)$ amounts to an element $M.N$ of $\Sigma^* x\colon A.B$ together with a proof that this element is free from the names in M.

The freefrom type in the elimination rule (Σ^*-E) can also be explained with this view of the $\Sigma^* x\colon A.B$ as a type of pairs with hidden/bound names. An element M of the type $\Sigma^* x\colon A.B$ represents a pair with hidden/bound names. In order to access the components of this pair, we must deal with the hidden/bound names. As is common practice according to Barendregt's Variable Convention, we choose them freshly. This is formalised by the rule (Σ^*-E), which allows us to match M against the pair $x.y$ in the term $(\Gamma * x\colon A), y\colon B \vdash N : C^{*(x:A)}$. The context of this term contains the information that x is fresh for Γ, i.e. that all the hidden names are chosen freshly. Of course, we must make sure that the choice of fresh names has no side effect. This is the purpose of the freefrom type $C^{*(x:A)}$, which guarantees that N denotes an element of C that is free from the newly chosen names in x. Therefore, none of the hidden names can be revealed by the Σ^*-elimination.

The presence of freefrom types in the Σ^*-rules is also explained by the categorical structure of simple monoidal sums. There is a freefrom type in (Σ^*-I) because the term $x\colon A, y\colon B \vdash \mathrm{bind}_{(x:A)B}(x,y) : (\Sigma^* x\colon A.B)^{*(x:A)}$ corresponds to the unit $\eta_B\colon B \to W_A\Sigma_A^* B$ of the adjunction $\Sigma_A^* \dashv W_A$. There is a freefrom type in (Σ^*-E) because this rule derives from the adjoint correspondence of $\Sigma_A^* \dashv W_A$, as shown below.

$$(\Sigma^*\text{-E})\ \frac{(\Gamma * x\colon A), y\colon B \vdash N : C^{*(x:A)}}{\Gamma, z\colon \Sigma^* x\colon A.B \vdash \text{let } z \text{ be } x.y\colon \Sigma^* x\colon A.B \text{ in } N : C} \qquad \frac{B \to W_A C \quad \text{in } \mathbb{E}_{\Gamma * A}}{\Sigma_A^* B \to C \quad \text{in } \mathbb{E}_\Gamma}$$

The equations for Σ^*-types are complicated by the presence of freefrom types. However, the freefrom terms in the equations (Σ^*-β) and (Σ^*-η) are there only to use the functoriality of W_A. For normal weak Σ-types, such terms for W_A are not necessary, since in this case W_A agrees with the weakening functor. Indeed, if we replace the freefrom terms in (Σ^*-β) and (Σ^*-η) by weakening, i.e. remove all the freefrom terms, then we obtain instances of the standard β-η-equations for weak Σ-types. Semantically, the equations for Σ^* derive from the triangular identities for the adjunction $\Sigma_A^* \dashv W_A$.

It is important to note that the restriction to closed freefrom types has an impact on the expressivity of Σ^*-types. The freefrom type in the conclusion of (Σ^*-I), for instance, can be formed only if $\Sigma^* x\colon A.B$ is closed. We must therefore make the assumption $x\colon A \vdash B$ Type in this rule. Another point where the restriction to closed freefrom types has an effect is in the equations (Σ^*-β) and (Σ^*-η). For the terms on both sides of the equation in the conclusion of (Σ^*-β) to be typeable, we must restrict the context of R to $(x\colon A, y\colon B)$. We expect this to be a severe restriction with regards to provability of equations.

The above informal explanation of the elimination rule (Σ^*-E) may help to understand why in Chapter 3 the type $\Sigma^* x\colon A.B$ is constructed as a certain quotient of $\Sigma x\colon A.B$. Defining a function $f\colon (X/\!\sim) \to Y$ out of a quotient type is equivalent to defining a function $g\colon X \to Y$ on the representatives such that the choice of representative does not matter, i.e. that $x \sim x' \implies g(x) = g(x')$ holds. The elimination rule (Σ^*-E) allows us to define a function out of the quotient $\Sigma^* x\colon A.B$ by defining it on representatives. Given an equivalence class M in $\Sigma x\colon A.B$, we can assume a representative $x.y$ of M and use it in the definition of a function to C. The freefrom type ensures that the choice of representative does not matter. We also remark that in [56, §11] Jacobs constructs quotient types using a certain left adjoint. The definition of Σ^* as a left adjoint may well fit into that framework.

Some words are in order on why there are *strong* Π^*-types but no corresponding *strong* Σ^*-types. Why do we not need a rule like (Π^*-E-CGR$^+$) for the terms $\mathrm{bind}_{(x:A)B}(M,N)$? The reason is the restriction to closed freefrom types. The above Σ^*-rules with closed freefrom types are such that we can form $\mathrm{bind}_{(x:A)B}(M,N)$ only if $\Sigma^* x\colon A.B$ is a closed type. This makes it unnecessary to assume a rule similar to (Π^*-E-CGR$^+$). Indeed, if we restrict $\mathrm{app}^*_{(x:A)B}(M,N)$ so that $\Pi^* x\colon A.B$ must be a closed type, then (Π^*-E-CGR$^+$) becomes admissible. This can be shown by using (INJECT) together with the equality $\mathrm{app}^*_{(x:A)B}(M,N) = (\mathrm{let}\ M{*}N\ \mathrm{be}\ f{*}x\ \mathrm{in}\ \mathrm{app}^*_{(x:A)B}(f,x))$, which can be derived only if $\Pi^* x\colon A.B$ is closed. A similar argument applies to $\mathrm{bind}_{(x:A)B}(M,N)$. This explains why the restriction to closed freefrom types makes strong Σ^*-types unnecessary. Moreover, if we have open freefrom types then we do not need to assume Σ^*-types to be strong either, since open freefrom types make rules like (Π^*-E-CGR$^+$) derivable, as discussed in Section 8.1.3.

A general point that should be noted is that for closed types A and B, the types $\Sigma^* x\colon A.B$ and $A{*}B$ are *not* in general isomorphic. This is in contrast to Σ-types for which there always is an isomorphism $\Sigma x\colon A.B \cong A \times B$. An example showing that $\Sigma^* x\colon A.B$ and $A{*}B$ are not always isomorphic is given by $\Sigma^*_{\mathbf{N}}\mathbf{N}$ and $\mathbf{N}*\mathbf{N}$ in the Schanuel topos. Informally, the object $\Sigma^*_{\mathbf{N}}\mathbf{N}$ contains the α-equivalence classes of a name with respect to one name, i.e. an element of $\Sigma^*_{\mathbf{N}}\mathbf{N}$ is either a bound name or an unbound name. It can therefore be seen that $\Sigma^*_{\mathbf{N}}\mathbf{N}$ is isomorphic to $1+\mathbf{N}$, see [72] for a proof. Hence, there cannot be an isomorphism between $\Sigma^*_{\mathbf{N}}\mathbf{N}$ and $\mathbf{N}*\mathbf{N}$.

9.3 Interpretation

The interpretation of Σ^*-types is based on the adjunction $\Sigma^*_A \dashv W_A$ in the same way the interpretation of weak Σ-types is based on the adjunction $\Sigma_A \dashv \pi^*_A$.

Formation The type formation rule

$$(\Sigma^*\text{-TY})\ \frac{\Gamma * x\colon A \vdash B\ \mathrm{Type}}{\Gamma \vdash \Sigma^* x\colon A.B\ \mathrm{Type}}$$

is interpreted by an application of the functor Σ^*. The interpretation of the premise of this rule amounts

to an object B of $\mathbb{E}_{\Gamma * \{A\}}$, where A is an object of $\mathbb{E}_1$. The type $\Sigma^* x\colon A.B$ is then interpreted as the object $\Sigma^*_{\{A\}} B$ of $\mathbb{E}_\Gamma$.

Introduction

$$(\Sigma^*\text{-I})\ \frac{x\colon A \vdash B\ \text{Type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : B[M/x]}{\Gamma \vdash \text{bind}_{(x:A)B}(M,N) : (\Sigma^* x\colon A.B)^{*(M:A)}}$$

The interpretation of the premise $x\colon A \vdash B$ Type gives us an object A in $\mathbb{B}$ and an object B in $\mathbb{E}_{\{A\}}$. We interpret the term $(\Gamma * x\colon A), y\colon B \vdash \text{bind}_{(x:A)B}(x,y) : (\Sigma^* x\colon A.B)^{*(x:A)}$ by the unit $\eta_{\{A\}} : B \to W_{\{A\}}\Sigma^*_{\{A\}} B$ of the adjunction $\Sigma^*_{\{A\}} \dashv W_{\{A\}}$. The general interpretation of $\text{bind}_{(x:A)B}(M,N)$ follows from this by substitution. The restriction to closed freefrom types makes soundness straightforward to show. To show soundness as in Chapter 6, we need to show that $(\text{bind}_{(x:A)B}(M,N))[\sigma] = (\text{bind}_{(x:A)B}(M',N'))[\tau]$ follows (essentially) from $M[\sigma] = M'[\tau]$ and $N[\sigma] = N'[\tau]$. With the restriction to closed freefrom types, this follows from (INJECT) together with the equation $\text{bind}_{(x:A)B}(M,N) = (\text{let } M * N \text{ be } u * v \text{ in } \text{bind}_{(x:A)B}(u,v))$, which holds because we can form $\text{bind}_{(x:A)B}(M,N)$ only when the type $\Sigma^* x\colon A.B$ is closed.

Elimination The premises of the elimination rule

$$(\Sigma^*\text{-E})\ \frac{\Gamma \vdash M : \Sigma^* x\colon A.B \qquad (\Gamma * x\colon A), y\colon B \vdash N : C^{*(x:A)}}{\Gamma \vdash \text{let } M \text{ be } x.y\colon \Sigma^* x\colon A.B \text{ in } N : C}$$

amount to objects A and C of $\mathbb{E}_1$, an object B of $\mathbb{E}_{\Gamma * \{A\}}$, a morphism $M\colon 1_\Gamma \to \Sigma^*_{\{A\}} B$ in $\mathbb{E}_\Gamma$ and a morphism $N\colon B \to W_{\{A\}} !^*_\Gamma C$ in $\mathbb{E}_{\Gamma * \{A\}}$. Here, we write $!_\Gamma$ for the unique map of type $\Gamma \to 1$. By adjoint transposition for the adjunction $\Sigma^*_{\{A\}} \dashv W_{\{A\}}$ we obtain a morphism $N^\sharp\colon \Sigma^*_{\{A\}} B \to !^*_\Gamma C$ in $\mathbb{E}_\Gamma$. We take this morphism to be the interpretation of the term $\Gamma, z\colon \Sigma^* x\colon A.B \vdash \text{let } z \text{ be } x.y\colon \Sigma^* x\colon A.B \text{ in } N : C$. The interpretation of $(\text{let } M \text{ be } x.y\colon \Sigma^* x\colon A.B \text{ in } N)$ follows by substitution of M for z, i.e. is given by $N^\sharp \circ M\colon 1_\Gamma \to !^*_\Gamma C$. For reference in the interpretation of the equations below, we recall that the adjoint transpose $N^\sharp$ is defined to be $\varepsilon_{(!^*_\Gamma C)} \circ \Sigma^*_{\{A\}} N$, where ε is the counit of the adjunction $\Sigma^*_{\{A\}} \dashv W_{\{A\}}$.

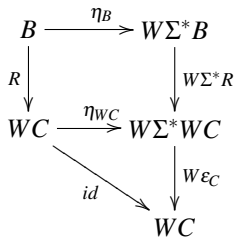
Equations The β and η-equations derive from the triangular identities for the adjunction $\Sigma^*_{\{A\}} \dashv W_{\{A\}}$. We first consider the following special case of $(\Sigma^*\text{-}\beta)$.

$$(\Sigma^*\text{-}\beta)\ \frac{x\colon A \vdash B\ \text{Type} \qquad x\colon A, y\colon B \vdash R : C^{*(x:A)}}{x\colon A, y\colon B \vdash \text{let } \text{bind}_{(x:A)B}(x,y) \text{ be } u^{*v}\colon (\Sigma^* x\colon A.B)^{*(x:A)} \text{ in } (\text{let } u \text{ be } x.y \text{ in } R)^{*v} = R : C^{*(x:A)}}$$

The general case of the rule $(\Sigma^*\text{-}\beta)$ follows from this special case by substituting M for x and N for y.

The premises of the above rule amount to objects A and C of $\mathbb{E}_1$, an object B of $\mathbb{E}_{\{A\}}$ and a morphism $R\colon B \to W_{\{A\}} C$ in $\mathbb{E}_{1 * \{A\}}$. We write short Σ^* for $\Sigma^*_{\{A\}}$ and W for $W_{\{A\}}$. The equation in the conclusion of the rule holds because the following diagram in $\mathbb{E}_{\Gamma * A}$ commutes for all objects Γ in $\mathbb{B}$. In fact, we only need the case where Γ is the terminal object 1, but we state the general case in order to make it easier to

see how to generalise it to open freefrom types.



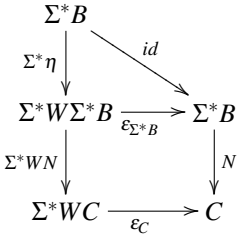
The square commutes by naturality of η, and the triangle is one of the triangular identities for $\Sigma^* \dashv W$.

The map $W\varepsilon_C \circ W\Sigma^* R \circ \eta_B$ in this diagram corresponds to the left term in the above equation. This can be seen by observing that η_B corresponds to $\mathrm{bind}_{(x:A)B}(x,y)$ and $\varepsilon_C \circ \Sigma^* R$ corresponds to (let u be $x.y$ in R). The use of freefrom types in the whole term amounts to the functoriality of W.

For the η-equation we also consider the unsubstituted case only. The general case follows by substitution.

$$(\Sigma^*\text{-}\eta)\ \frac{\vdash C\ \mathrm{Type} \qquad x\colon A \vdash B\ \mathrm{Type} \qquad z\colon \Sigma^* x\colon A.B \vdash N : C}{z\colon \Sigma^* x\colon A.B \vdash N = \mathrm{let}\ z\ \mathrm{be}\ x.y\ \mathrm{in}\ (\mathrm{let}\ \mathrm{bind}(x,y)\ \mathrm{be}\ z^{*x}\colon (\Sigma^* x\colon A.B)^{*(x:A)}\ \mathrm{in}\ N^{*x}) : C}$$

The premises of this rule amount to objects A and C of $\mathbb{E}_1$, an object B of $\mathbb{E}_{\{A\}}$ and a morphism $N\colon \Sigma^*_{\{A\}} B \to C$ in $\mathbb{E}_1$. We write short Σ^* for $\Sigma^*_{\{A\}}$ and W for $W_{\{A\}}$. The equation holds because the following diagram in $\mathbb{E}_\Gamma$ commutes for all objects Γ of $\mathbb{B}$.



The square in this diagram commutes by naturality of ε, and the triangle is one of the triangular identities for the adjunction $\Sigma^* \dashv W$.

9.4 Simple Monoidal Sums with Open Freefrom Types

In the above rules for Σ^*-types, we have had to make restrictions to ensure that all freefrom types are closed. If open freefrom types are available then these restrictions can be relaxed easily, and Σ^*-types can be formulated by the following rules. We omit the straightforward congruence rules, recalling from the discussion on Section 8.1.3 that with a good formulation of open freefrom types it is not necessary to assume non-standard congruence rules as those for strong Π^*-types.

Formation

$$(\Sigma^*\text{O-TY})\ \frac{\Gamma * x\colon A \vdash B\ \text{Type}}{\Gamma \vdash \Sigma^* x\colon A.B\ \text{Type}}$$

Introduction

$$(\Sigma^*\text{O-I})\ \frac{\Gamma * x\colon A \vdash B\ \text{Type} \qquad \Delta \vdash M : A \qquad \Gamma * \Delta \vdash N : B[M/x]}{\Gamma * \Delta \vdash \text{bind}_{(x:A)B}(M,N) : (\Sigma^* x\colon A.B)^{*(M:A)}}$$

Elimination

$$(\Sigma^*\text{O-E})\ \frac{\Gamma \vdash C\ \text{Type} \qquad \Gamma \vdash M : \Sigma^* x\colon A.B \qquad (\Gamma * x\colon A), y\colon B \vdash N : C^{*(x:A)}}{\Gamma \vdash \text{let } M \text{ be } x.y\colon \Sigma^* x\colon A.B \text{ in } N : C}$$

Equations

$$(\Sigma^*\text{O-}\beta)\ \frac{\begin{array}{c}\Gamma \vdash C\ \text{Type} \qquad \Delta \vdash M : A \\ \Gamma * x\colon A \vdash B\ \text{Type} \qquad \Gamma * \Delta \vdash N : B[M/x] \qquad (\Gamma * x\colon A), y\colon B \vdash R : C^{*(x:A)}\end{array}}{\Gamma * \Delta \vdash \begin{array}{l}\text{let bind}_{(x:A)B}(M,N) \text{ be } u^{*v}\colon (\Sigma^* x\colon A.B)^{*(M:A)} \text{ in } (\text{let } u \text{ be } x.y \text{ in } R)^{*v} \\ = R[M/x][N/y]\end{array} : C^{*(M:A)}}$$

$$(\Sigma^*\text{O-}\eta)\ \frac{\Gamma \vdash C\ \text{Type} \qquad \Gamma \vdash M : \Sigma^* x\colon A.B \qquad \Gamma, z\colon \Sigma^* x\colon A.B \vdash N : C}{\Gamma * \Delta \vdash N[M/z] = \text{let } M \text{ be } x.y \text{ in } (\text{let bind}(x,y) \text{ be } z^{*x}\colon (\Sigma^* x\colon A.B)^{*(x:A)} \text{ in } N^{*x}) : C}$$

9.5 Related and Further Work

As for related work, we are aware of only one type system containing a multiplicative version of Σ-types, namely FreshML 2000 with its abstraction types. In other work on multiplicative typing, as discussed in Section 4.5, only multiplicative function spaces and types similar to our $*$-types are studied. This does not cover Σ^*-types, since $*$-types are not a special case of Σ^*-types, as discussed in Section 9.2 above.

For future work, we believe that finding simpler formulations for Σ^*-types is most important. Due to the dependency of Σ^*-types on freefrom types, this also requires finding a simpler formulation for freefrom types. But perhaps it is also possible to find a formulation of Σ^*-types without freefrom types, possibly by using some external logic to establish freshness assertions rather than building these into the types via freefrom types. It would also be an interesting to find out is what the simplest possible type system containing usable Σ^*-types should be. Given our fibred formulation of Σ^* and the rules in this chapter, it appears likely that such a type-system should have dependent types of some form.

A more modest goal would be to find out whether Σ^*-types, which are a monoidal version of weak Σ-types, can be generalised to a monoidal version of strong Σ-types. For this we would have to find a certain isomorphism $(\Gamma * x\colon A), y\colon B \to \Gamma, z\colon \Sigma^* x\colon A.B$, and this would require to relax the rule (Σ^*O-E) so that C is allowed to depend on at least the term $\text{bind}(x,y)$.

Chapter 10

Categories with Bindable Names

In this chapter we now come to studying names and name-binding. The aim of this chapter is to give definitions that capture the concepts of names and name-binding. The main focus lies on capturing the concept of binding. The importance of binding is often emphasised in the literature. Miller & Tiu [73], for example, view binding as the central concept of their theory. Pitts [86] also emphasises the role of names as 'Names of entities that may be subject to binding by some of the syntactical constructions under consideration.' In this chapter we propose a general categorical definition of the notion of binding, as well as a more specialised notion of bindable names.

Our definition of binding is best explained by looking at the informal practice of working with α-equivalence classes of syntax terms. As explained in the introduction, abstract syntax involving variable binders is usually defined by first defining the raw syntax in which bound variables are concretely named and then taking the quotient of the raw syntax with respect to α-conversion, i.e. with respect to renaming of bound variables. However, when working with the abstract syntax, one does not normally use the α-equivalence classes. Instead, one works with concrete representatives of the α-equivalence classes. It is well-known that some care is required with the choice of the representatives, namely that the names chosen for the bound variables must be sufficiently fresh. This practice of working with freshly named instances of α-equivalence classes rather than with the equivalence classes themselves is known as the Barendregt Variable Convention (BVC), defined as follows in [9].

> 2.1.12. CONVENTION Terms that are α-congruent are identified. [...]
>
> 2.1.13. VARIABLE CONVENTION If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof) then in these terms all bound variables are chosen to be different from free variables. [...]
>
> 2.1.14. MORAL Using conventions 2.1.12 and 2.1.13 one can work with λ-terms in the naive way.

The success of informal work using the Barendregt Variable Convention suggests that it really does not matter whether we work with an α-equivalence classes or whether we work with freshly named instances. Some formal justification for this is provided by the work of Vestergaard and Brotherston [116].

The basis of our categorical definition of binding is that there is no essential difference between working with α-equivalence classes and working with freshly named instances. We want to formalise that working with a statement containing a bound variable is essentially the same as working with an instance of the statement at a fresh variable. First we need to define what we mean by variables and freshness. For the moment, we take a rather minimal approach to variables and freshness. We assume a category with an object V, which is thought of as the object of variables, and a monoidal structure $\otimes$, where $\Gamma \otimes V$ is thought of as a context Γ with an additional fresh variable. Next we need to define what we mean by statements containing a bound variable and the instantiation of such a bound variable. To this end, consider the internal type theory of a category with V and $\otimes$ as above. A binding operation should allow us to go from a statement in a context with an additional fresh variable $\Gamma \otimes V$ to a statement in context Γ by binding of the additional variable. In other words, we should be able to go from $\mathbb{B}/\Gamma \otimes V$ to $\mathbb{B}/\Gamma$. On the other hand, if in context Γ we have a statement with a bound variable then we should be able to instantiate it with a fresh variable. Since we can view a statement without a bound variable as one with a trivial binding, we should also be able to go from $\mathbb{B}/\Gamma$ to $\mathbb{B}/\Gamma \otimes V$. The intuition of binding and instantiation dictates that these two transitions should be inverses of each other. Furthermore, if, by analogy with the BVC, we want constructions with a bound variable to be essentially the same as constructions with the specific instances, we should require the two transitions to form an equivalence of categories. We make such an equivalence the definition of binding: We require an object V, a monoidal structure $\otimes$ and a certain equivalence of the slices $\mathbb{B}/\Gamma$ and $\mathbb{B}/\Gamma \otimes V$ for all Γ. In this chapter, we investigate some consequences of this definition and construct concrete instances of it.

10.1 Binding Structure

Our definition of binding is this:

Definition 10.1.1. A *category with binding structure* is a triple $(\mathbb{B}, \otimes, V)$ of a category $\mathbb{B}$ with pullbacks, a monoidal structure $\otimes$ on $\mathbb{B}$ and an object V of $\mathbb{B}$, for which the functor W_V defined by

$$\begin{array}{ccc} \mathbb{B}^{\to} & \xrightarrow{W_V} & \mathbb{B}/(-\otimes V) \\ & {\scriptstyle cod}\searrow \quad \swarrow{\scriptstyle Gl(-\otimes V)} & \\ & \mathbb{B} & \end{array} \qquad W_V : \begin{array}{ccc} B & \longmapsto & B \otimes V \\ f \downarrow & & f\otimes V \downarrow \\ \Gamma & & \Gamma \otimes V \end{array}$$

is an equivalence of fibrations.

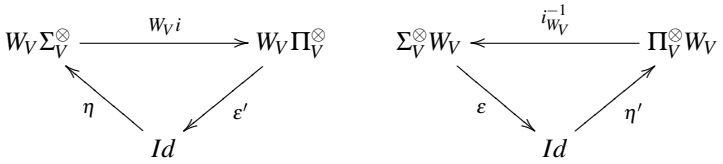
The requirement that $\mathbb{B}$ has pullbacks is necessary for the codomain fibration to exist. Note that in order for W_V to be an equivalence of fibrations it must necessarily be a fibred functor.

The content of this chapter revolves around the definition of a category with binding structure, which is why we have given this definition up front. However, even though the discussion in the previous section should give a first idea of how this definition can be understood, we still have to give a formal

justification as to why it captures binding. In the rest of this chapter we work towards this aim by investigating the abstract properties of categories with binding structure and by constructing concrete instances of these categories. We start by reformulating the definition of categories with binding structure in terms of more familiar constructs.

Proposition 10.1.2. *The following are equivalent.*

1. *The functor W_V is an equivalence of the fibrations.*
2. *The functor W_V has both a fibred left adjoint $\Sigma_V^{\otimes}$ and a fibred right adjoint $\Pi_V^{\otimes}$ and there exists a vertical (with respect to the codomain fibration) natural isomorphism $i\colon \Sigma_V^{\otimes} \to \Pi_V^{\otimes}$ such that the triangles*



   commute (note that this amounts to the four equations $id = \eta \circ \varepsilon' \circ W_V i$, $id = \varepsilon' \circ W_V i \circ \eta$, $id = \varepsilon \circ i_{W_V}^{-1} \circ \eta'$ and $id = i_{W_V}^{-1} \circ \eta' \circ \varepsilon$). Here, η and ε are the unit and counit of the adjunction $\Sigma_V^{\otimes} \dashv W_V$, and η' and ε' are the unit and counit of the adjunction $W_V \dashv \Pi_V^{\otimes}$.
3. *There exists a functor $\mathrm{H}\colon Gl(-\otimes V) \to cod$ that is both fibred left adjoint and fibred right adjoint to W_V. Moreover, the equations $\eta^{-1} = \varepsilon'$ and $\varepsilon^{-1} = \eta'$ hold, where η and ε are the unit and counit of the adjunction $\mathrm{H} \dashv W_V$, and η' and ε' are the unit and counit of the adjunction $W_V \dashv \mathrm{H}$.*

Proof.

1. $\Rightarrow$ 2. All the fibrations on base $\mathbb{B}$ and the fibred functors between them form a 2-category $Fib(\mathbb{B})$. It is a general 2-categorical fact that if W_V is an equivalence then it has both a fibred left adjoint $\Sigma_V^{\otimes}\colon Gl(-\otimes V) \to cod$ and a fibred right adjoint $\Pi_V^{\otimes}\colon Gl(-\otimes V) \to cod$ such that the unit and counit of both adjunctions are isomorphisms. Since both unit η' and counit ε' of the fibred adjunction $W_V \dashv \Pi_V^{\otimes}$ are isomorphisms, it follows that ε'^{-1} and η'^{-1} are the unit and counit of a fibred adjunction $\Pi_V^{\otimes} \dashv W_V$. The result now follows by uniqueness of adjoints. Define $i \stackrel{\text{def}}{=} \varepsilon_{\Pi_V^{\otimes}} \circ \Sigma_V^{\otimes}(\varepsilon'^{-1})\colon \Sigma_V^{\otimes} \to \Sigma_V^{\otimes} W_V \Pi_V^{\otimes} \to \Pi_V^{\otimes}$. It follows from naturality and the triangular identities for the two adjunctions that the triangles in the proposition commute. For example, we have

$$\begin{aligned}
\varepsilon' \circ W_V i \circ \eta &= \varepsilon' \circ W_V \varepsilon_{\Pi_V^{\otimes}} \circ W_V \Sigma_V^{\otimes}(\varepsilon'^{-1}) \circ \eta && \text{by definition} \\
&= \varepsilon' \circ W_V \varepsilon_{\Pi_V^{\otimes}} \circ \eta_{W_V \Pi_V^{\otimes}} \circ \varepsilon'^{-1} && \text{by naturality of } \eta \\
&= \varepsilon' \circ \varepsilon'^{-1} && \text{by triangular identity} \\
&= id.
\end{aligned}$$

2. $\Rightarrow$ **3.** Let $\mathrm{H} \stackrel{\text{def}}{=} \Sigma_V^{\otimes}$ (just as well $\mathrm{H} \stackrel{\text{def}}{=} \Pi_V^{\otimes}$ is possible). With this definition we get a fibred adjunction $\mathrm{H} \dashv W_V$ with unit η and counit ε, as well as a fibred adjunction $W_V \dashv \mathrm{H}$ with unit $i_{W_V}^{-1} \circ \eta'$ and counit $\varepsilon' \circ W_V i$. The required equations are now exactly given by the commuting triangles.

3. $\Rightarrow$ **1.** By assumption, both η and ε vertical isomorphisms, which implies that W_V is part of a fibred adjoint equivalence.

□

The proposition connects a binding structure with the simple monoidal sums $\Sigma^{\otimes}$ and simple monoidal products $\Pi^{\otimes}$ studied in previous chapters. It shows that binding is just a special property of $\Sigma_V^{\otimes}$ and $\Pi_V^{\otimes}$. We will use the description of binding in terms of simple monoidal sums and products to add binding to type theory. An intuitive description of the meaning of the triangles in the proposition is given for the Schanuel topos in Section 10.3.1 below.

It should be clear that the above proposition allows us to generalise the concept of binding structure to fibrations other than the codomain fibration. Once we have explained W_V, $\Sigma^{\otimes}$ and $\Pi^{\otimes}$ for a fibration, the proposition gives us a notion of binding structure for that fibration. Furthermore, while in the definition of binding structure we have taken the codomain fibration, which corresponds to the internal language of the category, it should be possible to take other fibrations, such as subobject fibrations. If, for example, we take the subobject fibration over the Schanuel topos and take $V \stackrel{\text{def}}{=} \mathbf{N}$ and $\otimes \stackrel{\text{def}}{=} *$, then the resulting binding structure amounts to the freshness quantifier of Gabbay & Pitts.

We consider some consequences of the definition of binding structure.

Proposition 10.1.3. *In any category with binding structure $(\mathbb{B}, \otimes, V)$, the functor $- \otimes V$ has a right adjoint $V \multimap -$. Furthermore, if $\mathbb{B}$ is cartesian closed then the monoidal exponent $V \multimap -$ has a further right adjoint.*

Proof. The existence of the right adjoint $V \multimap -$ follows in the same way as monoidal closure follows in the proof of Proposition 2.5.6, since the functor $- \otimes V$ can be written as the lower composite in the chain below.

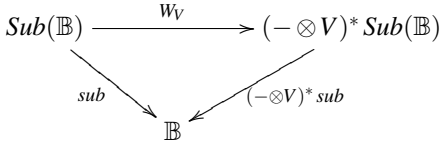
$$\mathbb{B} \underset{\cong}{\overset{\cong}{\leftarrow \top \rightarrow}} \mathbb{B}/1 \underset{W_V}{\overset{\Pi_V^{\otimes}}{\leftarrow \top \rightarrow}} \mathbb{B}/(1 \otimes V) \underset{\Sigma_!}{\overset{!^*}{\leftarrow \top \rightarrow}} \mathbb{B}/1 \underset{\cong}{\overset{\cong}{\leftarrow \top \rightarrow}} \mathbb{B}$$

The functor $V \multimap -$ is then taken as the upper composite in this diagram. If $\mathbb{B}$ is cartesian closed then the codomain fibration has simple products, which implies that we have an adjunction $!^* \dashv \Pi_!$. Furthermore, we have $\Pi_V^{\otimes} \cong \Sigma_V^{\otimes} \dashv W_V$. From these two facts, if follows that the functor $V \multimap -$ has a further right adjoint, given by upper row in the diagram below.

$$\mathbb{B} \underset{\cong}{\overset{\cong}{\rightarrow \top \leftarrow}} \mathbb{B}/1 \underset{\Pi_V^{\otimes}}{\overset{W_V}{\rightarrow \top \leftarrow}} \mathbb{B}/(1 \otimes V) \underset{!^*}{\overset{\Pi_!}{\rightarrow \top \leftarrow}} \mathbb{B}/1 \underset{\cong}{\overset{\cong}{\rightarrow \top \leftarrow}} \mathbb{B}$$

□

Any category with binding structure $(\mathbb{B},\otimes,V)$ has a quantifier similar to the freshness quantifier of Gabbay & Pitts. To see this, consider the subobject logic of $\mathbb{B}$, i.e. the subobject fibration $sub\colon\ Sub(\mathbb{B})\to\mathbb{B}$. Since W_V is a fibred functor it preserves pullbacks, which implies that W_V preserves monomorphisms. Therefore, W_V restricts to a fibred functor



Proposition 10.1.4. *For any category with binding structure $(\mathbb{B},\otimes,V)$ there exists a fibred functor $\forall$ of type $(-\otimes V)^*\,sub\to sub$ that is both fibred left and fibred right adjoint to $W_V\colon\ sub\to(-\otimes V)^*\,sub$.*

Proof. Since $\Pi_V^\otimes$ is a right adjoint it preserves the terminal object and monomorphisms. In particular, for a monomorphism $m\colon A\rightarrowtail\Gamma\otimes V$, the action of $\Pi_V^\otimes$ on the morphism in $\mathbb{B}/(\Gamma\otimes V)$ in the diagram on the left below gives a morphism in $\mathbb{B}/\Gamma$ as in the diagram on the right below.

Therefore, $\Pi_V^\otimes$ maps any object m in $\mathbb{B}/(\Gamma\otimes V)$ that is given by a monomorphism to an object in $\mathbb{B}/\Gamma$ that is also given by a monomorphism. By $\Sigma_V^\otimes\cong\Pi_V^\otimes$, this also holds for $\Sigma_V^\otimes$. Hence, both $\Sigma^\otimes$ and $\Pi^\otimes$ restrict to endofunctors on $Sub(\mathbb{B})$. We write $\exists^\otimes$ and $\forall^\otimes$ for the restrictions.

Since the morphisms between two subobjects in $Sub(\mathbb{B})$ are in one-to-one correspondence with morphisms in $\mathbb{B}^\to$ between monomorphisms representing the subobjects, it follows that the fibred adjunctions $\Sigma_V^\otimes\vdash W_V\vdash\Pi_V^\otimes$ between cod and $Gl(-\otimes V)$ restrict to fibred adjunctions $\exists_V^\otimes\vdash W_V\vdash\forall_V^\otimes$ between sub and $(-\otimes V)^*\,sub$.

From the natural vertical isomorphism $\Sigma^\otimes\cong\Pi^\otimes$ it follows that we have a natural vertical isomorphism $\exists^\otimes\cong\forall^\otimes$. Since the fibres of sub are partial orders, this implies $\exists^\otimes=\forall^\otimes$, and we write $\forall$ for this functor. □

Spelling out the proposition in non-fibred terms, we get that, for each object Γ in $\mathbb{B}$, there is a functor $\forall\colon\ Sub(\Gamma\otimes V)\to Sub(\Gamma)$ that is both left and right adjoint to $W_V\colon\ Sub(\Gamma)\to Sub(\Gamma\otimes V)$ and that satisfies the Beck-Chevalley condition $\forall(u\otimes V)^*=u^*\forall$ for all morphisms u in $\mathbb{B}$.

The monoidal exponent $V\multimap-$ and the quantifier $\forall$ are very well-behaved.

Proposition 10.1.5. *Let $(\mathbb{B},\otimes,V)$ be a category with binding structure that has finite products, finite*

coproducts and that is cartesian closed. Then, for all objects A and B in $\mathbb{B}$, there are isomorphisms

$$\begin{aligned} V \multimap 1 &\cong 1 \\ V \multimap (A \times B) &\cong (V \multimap A) \times (V \multimap B) \\ V \multimap 0 &\cong 0 \\ V \multimap (A + B) &\cong (V \multimap A) + (V \multimap B) \\ V \multimap (A \Rightarrow B) &\cong (V \multimap A) \Rightarrow (V \multimap B), \end{aligned}$$

in which $V \multimap -$ is the monoidal exponent from Proposition 10.1.3. Moreover, these isomorphisms are natural in A and B.

Proof. The first four cases follow immediately from Proposition 10.1.3, because the functor $V \multimap -$ has both left and right adjoint and therefore preserves limits and colimits. For the last case, note that $\Pi_V^{\otimes} : \mathbb{B}/(1 \otimes V) \to \mathbb{B}/1$ must preserve all exponents that exist in $\mathbb{B}/(1 \otimes V)$, since $\Pi_V^{\otimes}$ is an equivalence of categories. By the definition of $V \multimap -$ in the proof of Proposition 10.1.3, the object $V \multimap (A \Rightarrow B)$ is defined to be $\Pi_V^{\otimes} !^*(A \Rightarrow B)$, where $! : 1 \otimes V \to 1$ is the terminal map and where we have left implicit the isomorphism $\mathbb{B}/1 \cong \mathbb{B}$. It is a standard result, see e.g. [57, Lemma 1.5.2], that $!^*$ preserves exponents, i.e. the exponent $(!^*A) \Rightarrow (!^*B)$ exists in $\mathbb{B}/(1 \otimes V)$ and is isomorphic to $!^*(A \Rightarrow B)$. This gives us natural isomorphisms

$$\begin{aligned} V \multimap (A \Rightarrow B) &\cong \Pi_V^{\otimes} !^*(A \Rightarrow B) \\ &\cong \Pi_V^{\otimes} ((!^*A) \Rightarrow (!^*B)) \\ &\cong (\Pi_V^{\otimes} !^*A)) \Rightarrow (\Pi_V^{\otimes} !^*B) \\ &\cong (V \multimap A) \Rightarrow (V \multimap B), \end{aligned}$$

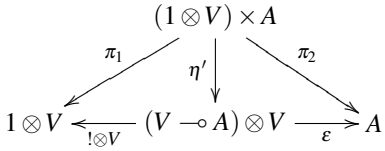
thus completing the proof. □

Proposition 10.1.6. *In any coherent category with binding structure $(\mathbb{B}, \otimes, V)$ the following equalities of subobjects hold.*

$$\begin{aligned} И \top &= \top \\ И(\varphi \wedge \psi) &= (И\varphi) \wedge (И\psi) \\ И \bot &= \bot \\ И(\varphi \vee \psi) &= (И\varphi) \vee (И\psi) \\ И(\varphi \supset \psi) &= (И\varphi) \supset (И\psi) \end{aligned}$$

Proof. By $И \dashv W_V \dashv И$ and because $Sub(\Gamma)$ is a partial order, the unit $\varphi \leq И W_V \varphi$ of the right adjunction and the counit $И W_V \varphi \leq \varphi$ of the left adjunction imply $\varphi = И W_V \varphi$ for all φ in $Sub(\Gamma)$. Dually, we get $\psi = W_V И \psi$ for all ψ in $Sub(\Gamma \otimes V)$. Therefore, the units and counits of both adjunctions are isomorphisms, which means that W_V and $И$ constitute an equivalence of the categories $Sub(\Gamma)$ and $Sub(\Gamma \otimes V)$. The assertion follows since an equivalence always preserves limits, colimits and exponents. □

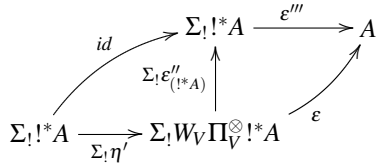
Proposition 10.1.7. *Let $(\mathbb{B},\otimes,V)$ be a category with binding structure, where $\mathbb{B}$ has finite products. For each object A there exists a map η' making the following diagram commute.*



Proof. In essence, the map η' is the unit of the fibred adjunction $\Sigma_V^{\otimes}\dashv W_V$, the left triangle commutes because the unit of a fibred adjunction is vertical, and the right triangle commutes because the left triangle in Proposition 10.1.2 commutes. Given this, the rest of the proof is little more than unwinding of definitions.

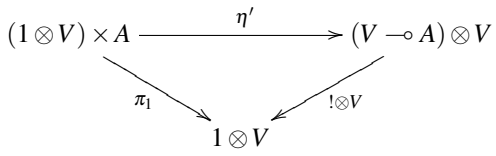
Without loss of generality we assume $\Sigma_V^{\otimes}=\Pi_V^{\otimes}$ and that the isomorphism $i\colon \Sigma_V^{\otimes}\to\Pi_V^{\otimes}$ from Proposition 10.1.2 is the identity. Write η' and ε' for the unit and counit of $\Sigma_V^{\otimes}\dashv W_V$, write η'' and ε'' for the unit and counit of $W_V\dashv\Pi_V^{\otimes}$ and write η''' and ε''' for the unit of the adjunction $\Sigma_!\dashv !^*$ in which $!$ is the terminal map $!\colon 1\otimes V\to 1$. In the rest of the proof we make the isomorphism between the categories $\mathbb{B}$ and $\mathbb{B}/1$ implicit.

By construction of $V\multimap -$ in Proposition 10.1.3, we have $(-\otimes V)=\Sigma_! W_V$ and $(V\multimap -)=\Pi_V^{\otimes}!^*$ and the counit ε of the adjunction $-\otimes V\dashv V\multimap -$ is $\varepsilon'''\circ\Sigma_!\varepsilon''_{!^*}$. Therefore, the right-hand triangle in following diagram in $\mathbb{B}/1$ commutes by definition. The left-hand triangle commutes because the left triangle in Proposition 10.1.2 commutes.



(10.1)

Now let A be an object of $\mathbb{B}/1$. We unfold the type of the unit $\eta'\colon !^*A\to W_V\Sigma_V^{\otimes}!^*A$ in $\mathbb{B}/(1\otimes V)$. Notice that its codomain is $W_V(V\multimap A)$. The map η' therefore fits in the following commuting diagram.



This gives the left triangle in the diagram of the proposition.

It remains to show that the right triangle in the triangle of the proposition commutes as well. Since $\Sigma_!$ acts by post-composition on objects and as the identity on morphisms, the map $\varepsilon\circ\eta'$ in $\mathbb{B}$ can be identified with the map $\varepsilon\circ\Sigma_!\eta'$ in $\mathbb{B}/1$. By the diagram (10.1), this equals ε'''. Finally, the counit ε''' is easily be seen to be the second projection $(1\otimes V)\times A\to A$. We have therefore shown the equality $\varepsilon\circ\eta'=\pi_1\colon V\times A\to A$, which is just what is required to make the right triangle of the diagram in the proposition commute. □

Given the above propositions, it might be interesting to ask for an axiomatisation of a category with binding structure $(\mathbb{B}, \otimes, V)$ in terms of only the monoidal structure $\otimes$ and the monoidal exponent $V \multimap -$. At present, we do not know of such a restatement of binding structure in terms of the monoidal structure. Perhaps the closest to such a restatement is Menni's axiomatisation of binding [72] that we discuss in the next section. On the other hand, we prefer the definition of binding structure in terms of fibrations as it is both concise and has a natural intuitive explanation.

10.2 Categories with Bindable Names

In the definition of a category with binding structure $(\mathbb{B}, \otimes, V)$ we have made very few assumptions on V and $\otimes$, allowing V to be an arbitrary object and $\otimes$ to be an arbitrary monoidal structure. As a consequence, V and $\otimes$ need not correspond to the intuition of an object of variables and a freshness relation. For instance, the triple $(\mathbb{B}, \times, 1)$ is a trivial binding structure for any cartesian category $\mathbb{B}$.

In this section we define categories with bindable names as those categories with binding structure $(\mathbb{B}, *, \mathbf{N})$ in which the object $\mathbf{N}$ is like an object of names and the monoidal structure $*$ is like a freshness relation for those names. The axioms for names and freshness are derived from the properties of the Schanuel topos $\mathbb{S}$, as it has been argued convincingly by Gabbay & Pitts [38] that the object $\mathbf{N}$ and the monoidal structure $*$ in $\mathbb{S}$ capture a good notion of names and freshness. In essence, we require the object of names to be a decidable object and $*$ to be a binary relation extending the inequality relation on $\mathbf{N}$ in a reasonable way.

Definition 10.2.1 (Category with Bindable Names)**.** A *category with bindable names* is a category with binding structure $(\mathbb{B}, *, \mathbf{N})$ satisfying the following additional properties.

1. $\mathbb{B}$ is a coherent category.

2. The monoidal structure $*$ is a strict affine symmetric monoidal structure and, for each monomorphism $m\colon B \rightarrowtail \Gamma$, the following square is a pullback

$$\begin{array}{ccc} B * \mathbf{N} & \xrightarrow{\dot{\pi}_1} & B \\ {\scriptstyle m*\mathbf{N}}\downarrow & \lrcorner & \downarrow{\scriptstyle m} \\ \Gamma * \mathbf{N} & \xrightarrow[\dot{\pi}_1]{} & \Gamma \end{array}$$

3. For the object $\mathbf{N}$, the canonical inclusion $\iota\colon \mathbf{N} * \mathbf{N} \rightarrowtail \mathbf{N} \times \mathbf{N}$ is the complement of the diagonal $\delta\colon \mathbf{N} \rightarrowtail \mathbf{N} \times \mathbf{N}$.

We emphasise that the purpose of this definition is to axiomatise names and freshness, not just to exclude trivial instance of categories with binding structure. There are interesting categories with binding structure, such as $(\mathbf{Sets}^{\mathbb{P}}, \otimes, X)$ discussed in Section 10.3.3, which could not be reasonably called a category with bindable names. Furthermore, the definition is by no means forced. It can be varied to

include more or less information about the properties of names. For example, in the Schanuel topos it should be possible to use swapping to add more information about names.

In the rest of this section we explain the axioms in the definition of bindable names and consider some of its consequences. The requirement for $\mathbb{B}$ to be a coherent category is there to give us a reasonable internal logic in which we can state and work with the properties of names. The assumptions on $*$ form a useful set of properties of freshness. That the monoidal structure is strict affine, i.e. that all canonical maps $A * B \rightarrowtail A \times B$ are monomorphic, makes available that the monoidal structure is a binary relation in the subobject logic. The additional pullback corresponds to the intuition that an element b of B is fresh for some name n in $\mathbf{N}$ if and only if n is fresh for $m(b) \in \Gamma$. This corresponds to Proposition 3.3.5.2 in Chapter 3. Finally, the third point of the definition relates the object of names and the freshness relation by declaring that two names are different exactly when they are fresh.

We state some properties of the object of names.

Proposition 10.2.2. *The following are true in any category with bindable names* $(\mathbb{B}, *, \mathbf{N})$.

1. *The following square is both a pullback and a pushout*

$$\begin{array}{ccc} 0 & \longrightarrow & \mathbf{N} \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \delta} \\ \mathbf{N} * \mathbf{N} & \xrightarrow[\iota]{} & \mathbf{N} \times \mathbf{N} \end{array}$$

2. *The morphism* $[\delta, \iota] \colon \mathbf{N} + (\mathbf{N} * \mathbf{N}) \to (\mathbf{N} \times \mathbf{N})$*, where* δ *is the diagonal map and* ι *is the canonical inclusion for* $*$*, is an isomorphism.*

Proof.

1. By assumption, the subobjects ι and δ complement each other. By definition, this means that their intersection is 0 and their union is $\mathbf{N} \times \mathbf{N}$. The assertion then follows as in [57, Prop. 1.4.3].

2. By the first point, using the fact that the pushout under 0 is a coproduct.

□

Proposition 10.2.3. *In any category with bindable names* $(\mathbb{B}, *, \mathbf{N})$*, the object* $\mathbf{N}$ *is an infinite object in the following sense: For any number of generalised elements* $x_1, \ldots, x_n \colon \Gamma \to \mathbf{N}$ *there exists an epimorphism* $p \colon \Delta \twoheadrightarrow \Gamma$ *and a generalised element* $y \colon \Delta \to \mathbf{N}$ *such that, for any* i *with* $1 \leq i \leq n$*, the equaliser of* y *and* $x_i \circ p$ *is* 0*.*

Proof. First we show that, for each object Γ, the projection $\pi_1 \colon \Gamma * \mathbf{N} \to \Gamma$ is an epimorphism. Since $\mathbb{B}$ is a coherent category it has images, i.e. each morphism f factors into a cover c followed by a monomorphism i. The subobject classified by i is called the image of f. A cover is a map with the property

that, whenever $c = m \circ c'$ holds for a monomorphism m, then m must be an isomorphism. In a regular category each cover is an epimorphism. The image of $f\colon A \to B$ can be constructed by applying the existential quantifier $\exists_f\colon \mathit{Sub}(A) \to \mathit{Sub}(B)$ to the true proposition $\top_A$, i.e. the terminal object in $\mathit{Sub}(A)$. To show that $\dot{\pi}_1 : \Gamma * \mathbf{N} \to \Gamma$ is an epimorphism, it suffices to show that its image is the true proposition, since then $\dot{\pi}_1$ factors into an epi followed by an isomorphism. Therefore, it suffices to show $\top_\Gamma \leq \exists_{\dot{\pi}_1}(\top_{\Gamma * \mathbf{N}})$. To show this we observe that, by the diagram in Definition 10.2.1.2, we have an equality of functors $\dot{\pi}_1^* = W_{\mathbf{N}}\colon \mathit{Sub}(\Gamma) \to \mathit{Sub}(\Gamma * \mathbf{N})$. By uniqueness of adjoints and Proposition 10.1.4, this implies $\forall = \exists_{\dot{\pi}_1} = \forall_{\dot{\pi}_1}$. Using this and adjoint transposition, we get

$$\dfrac{\dfrac{\dot{\pi}_1^* \top_\Gamma = \top_{\Gamma * \mathbf{N}} \leq \top_{\Gamma * \mathbf{N}} \quad \text{in } \mathit{Sub}(\Gamma * \mathbf{N})}{\top_\Gamma \leq \forall_{\dot{\pi}_1}(\top_{\Gamma * \mathbf{N}}) \quad \text{in } \mathit{Sub}(\Gamma)}}{\top_\Gamma \leq \exists_{\dot{\pi}_1}(\top_{\Gamma * \mathbf{N}}) \quad \text{in } \mathit{Sub}(\Gamma)}$$

which completes the proof that $\dot{\pi}_1 : \Gamma * \mathbf{N} \to \Gamma$ is an epimorphism.

To show that $\mathbf{N}$ is an infinite object, suppose we are given elements $x_1, \ldots, x_n\colon \Gamma \to \mathbf{N}$. Define p to be the map $\dot{\pi}_1 : \Gamma * \mathbf{N} \twoheadrightarrow \Gamma$, which we have just shown to be epimorphic, and let y be the second projection $\dot{\pi}_2\colon \Gamma * \mathbf{N} \to \mathbf{N}$. To show that, for $1 \leq i \leq n$, the equaliser of y and $x_i \circ p$ is an initial object, it suffices to show that

$$0 \longrightarrow \mathbf{N} * \mathbf{N} \underset{\dot{\pi}_2}{\overset{\dot{\pi}_1}{\rightrightarrows}} \mathbf{N}$$

is an equaliser diagram. This is the case because if we have any map $u\colon X \to \Gamma * \mathbf{N}$ equalising y and $x_i \circ p$ then $(x_i * \mathbf{N}) \circ u$ equalises $\dot{\pi}_1, \dot{\pi}_2\colon \mathbf{N} * \mathbf{N} \to \mathbf{N}$, so that using the above equaliser we get a map from X to 0, and in a coherent category this implies that X is also initial, see [57, Lemma 1.4.1]. That the above is an equaliser follows from the pullback in Proposition 10.2.2.1, since, for any map $u\colon X \to \mathbf{N} * \mathbf{N}$ with $\dot{\pi}_1 \circ u = \dot{\pi}_2 \circ u$, we have $\iota \circ u = \delta \circ (\dot{\pi}_1 \circ u)$, so that the pullback gives the required map $X \to 0$. □

The previous proposition can also be formulated in the coherent logic of $\mathbb{B}$. That $\mathbf{N}$ is an infinite object then means that the following sequent, in which we write $x\#y$ for the proposition represented by the canonical monomorphism $A * B \rightarrowtail A \times B$, holds for all $n \geq 0$.

$$x_1 : \mathbf{N}, \ldots, x_n : \mathbf{N} \mid \top \vdash \exists y : \mathbf{N}.\, (x_1 \# y) \wedge \cdots \wedge (x_n \# y)$$

Proposition 10.2.4. *A topos with binding structure $(\mathbb{B}, *, \mathbf{N})$ is a category with bindable names if and only if the following two conditions hold.*

- *2′. The monoidal structure $*$ is a strict affine symmetric monoidal structure such that the functor $(-) * \mathbf{N}$ preserves pullbacks.*
- *3. The morphism $[\delta, \iota]\colon \mathbf{N} + (\mathbf{N} * \mathbf{N}) \to (\mathbf{N} \times \mathbf{N})$, where δ is the diagonal map and ι is the canonical inclusion for $*$, is an isomorphism.*

Proof. Condition 1 in the definition of bindable names is satisfied since any topos is a coherent category. For condition 2, we have to show that, for any monomorphism $m\colon B \rightarrowtail \Gamma$, the diagram in point 2 of the definition of bindable names is a pullback. Since $\mathbb{B}$ is a topos, for any monomorphism $m\colon B \rightarrowtail \Gamma$ there exists a characteristic map χ such that the following square is a pullback.

$$\begin{array}{ccc} B & \xrightarrow{!_B} & 1 \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle true} \\ \Gamma & \xrightarrow[\chi]{} & \Omega \end{array}$$

Since the functor $- * \mathbf{N}$ preserves pullbacks, the left square in the diagram below is a pullback as well.

$$\begin{array}{ccccc} B*\mathbf{N} & \xrightarrow{!_B*\mathbf{N}} & 1*\mathbf{N} & \xrightarrow{!_{1*\mathbf{N}}} & 1 \\ {\scriptstyle m*\mathbf{N}}\downarrow & \lrcorner & \downarrow{\scriptstyle true*\mathbf{N}} & & \downarrow{\scriptstyle true} \\ \Gamma*\mathbf{N} & \xrightarrow[\chi*\mathbf{N}]{} & \Omega*\mathbf{N} & \xrightarrow[\dot{\pi}_1]{} & \Omega \end{array}$$

We show directly that the square on the right in this diagram is also a pullback. To see this, assume $f\colon \Gamma \to \Omega * \mathbf{N}$ and $g\colon \Gamma \to 1$ with $\dot{\pi}_1 \circ f = true \circ g$. As the pullback mediator $h\colon \Gamma \to 1 * \mathbf{N}$, we can take $h = (!_\Omega * \mathbf{N}) \circ f$. Since $g = !_\Gamma$, by the universal property of the terminal object, it suffices to show $(true * \mathbf{N}) \circ h = f$. This follows from the equations

$$\dot{\pi}_1 \circ (true * \mathbf{N}) \circ h = \dot{\pi}_1 \circ (true * \mathbf{N}) \circ (!_\Omega * \mathbf{N}) \circ f = true \circ !_\Omega \circ \dot{\pi}_1 \circ f = true \circ !_\Gamma = \dot{\pi}_1 \circ f$$

$$\dot{\pi}_2 \circ (true * \mathbf{N}) \circ h = \dot{\pi}_2 \circ (true * \mathbf{N}) \circ (! * \mathbf{N}) \circ f = \dot{\pi}_2 \circ f$$

by using the fact that $\langle \dot{\pi}_1, \dot{\pi}_2 \rangle\colon \Omega * \mathbf{N} \rightarrowtail \Omega \times \mathbf{N}$ is monomorphic. Showing uniqueness is straightforward.

To show that the diagram in point 2 of the definition of bindable names is a pullback, we have to show that the left square in

$$\begin{array}{ccccc} B*\mathbf{N} & \xrightarrow{\dot{\pi}_1} & B & \xrightarrow{!_B} & 1 \\ {\scriptstyle m*\mathbf{N}}\downarrow & & \downarrow{\scriptstyle m} & \lrcorner & \downarrow{\scriptstyle true} \\ \Gamma*\mathbf{N} & \xrightarrow[\dot{\pi}_1]{} & \Gamma & \xrightarrow[\chi]{} & \Omega \end{array}$$

is a pullback. By the pullback lemma, it suffices to show that the outer square in this diagram is a pullback. But since both $\chi \circ \dot{\pi}_1 = \dot{\pi}_1 \circ (\chi * \mathbf{N})$ and $!_B \circ \dot{\pi}_1 = !_{1*\mathbf{N}} \circ (!_B * \mathbf{N})$ hold, we have already shown this above. This shows the diagram in point 2 of the definition of bindable names to be a pullback.

Finally, the third condition in the definition of categories with bindable names holds because in a topos coproducts are disjoint. □

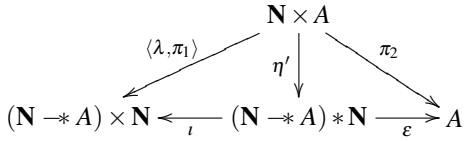
In any category with bindable names $(\mathbb{B}, *, \mathbf{N})$, the functor $(-) * \mathbf{N}$ has, by Proposition 10.1.3, a right adjoint which we denote by $\mathbf{N} \multimap (-)$. The quantifier $И$ from Proposition 10.1.4 can in a category with bindable names be expressed in terms of ordinary quantification.

Proposition 10.2.5. *For a category with bindable names* $(\mathbb{B}, *, \mathbf{N})$ *there is a fibred functor* $\forall$ *of type* $(- * \mathbf{N})^* sub_{\mathbb{B}} \to sub_{\mathbb{B}}$ *that is both fibred left and fibred right adjoint to the fibred functor* $\dot{\pi}_1^*$ *of type* $sub_{\mathbb{B}} \to (- * \mathbf{N})^* sub_{\mathbb{B}}$ *given by reindexing along the projection* $\dot{\pi}_1 \colon (-) * \mathbf{N} \to (-)$.

Proof. By Proposition 10.1.4 and the equality $W_{\mathbf{N}} = \dot{\pi}_1^*$, which follows from the diagram in Definition 10.2.1.2. □

We end this section by relating categories with bindable names to Menni's axiomatisation of binding [72]. First, it is easily seen that the functor $- * \mathbf{N}$ and the natural transformation $\dot{\pi}_1 \colon (-) * \mathbf{N} \to (-)$ form a $\forall$-relation in the sense of [72], meaning that $(-) * \mathbf{N}$ preserves pullbacks, that, for all objects Γ, the map $\dot{\pi}_1^* \colon Sub(\Gamma) \to Sub(\Gamma * \mathbf{N})$ is an isomorphism and that the map $\langle ! * \mathbf{N}, \dot{\pi}_1 \rangle \colon \Gamma * \mathbf{N} \to (1 * \mathbf{N}) \times \Gamma$ is a monomorphism. The following proposition therefore shows that Menni's binders, corresponding to the map λ in the proposition, are implied by our definition.

Proposition 10.2.6. *Let* $(\mathbb{B}, *, \mathbf{N})$ *be a category with bindable names. For each object* A *there exists a map* $\lambda \colon \mathbf{N} \times A \to (\mathbf{N} \multimap A)$ *such that there exists a map* $\eta' \colon \mathbf{N} \times A \to (\mathbf{N} \multimap A) * \mathbf{N}$ *making the following diagram commute.*



Proof. This follows directly from Proposition 10.1.7 using the fact that $*$ is a strict affine monoidal structure. We use Proposition 10.1.7 to obtain η' and define $\lambda = \dot{\pi}_1 \circ \eta'$. □

Menni observes that there is a relation between so-called pre-binders and a right adjoint to $\mathbf{N} \multimap (-)$. The precise role of this right adjoint nevertheless remains 'slightly mysterious'. With our definition of binding structure, the right adjoint arises naturally from the isomorphism $\Sigma_{\mathbf{N}}^* \cong \Pi_{\mathbf{N}}^*$. We have seen in the proof of Proposition 10.1.3, that the monoidal exponent $\mathbf{N} \multimap (-)$ can be regarded as a special case of $\Pi_{\mathbf{N}}^*$, and that it therefore follows from $\Pi_{\mathbf{N}}^* \cong \Sigma_{\mathbf{N}}^* \dashv W_{\mathbf{N}}$ that $\mathbf{N} \multimap (-)$ has a right adjoint. Moreover, it can be seen from the construction of the right adjoint to $\mathbf{N} \multimap (-)$ in Proposition 10.1.3, that the right adjoint can be expressed in the syntax of the type theory. It maps a type $(\vdash B \text{ Type})$ to the type $(\vdash \Pi n \colon \mathbf{N}. B^{*(n:\mathbf{N})} \text{ Type})$.

Having shown that any category with bindable names implies Menni's binders, it is natural to ask if the converse is also true. In a boolean topos, this is indeed the case. Let $\mathbb{B}$ be a boolean topos with an object $\mathbf{N}$ and a strict affine monoidal structure $*$. In terms of this data, Menni's definitions of $\forall$-relation and binders amount to the properties stated in Propositions 10.2.5 and 10.2.6 in addition to $(-) * \mathbf{N}$ preserving pullbacks. In the rest of this section, we show that this structure is enough to make $(\mathbb{B}, *, \mathbf{N})$ a category with bindable names. We remark that Menni only makes use of a special case of the monoidal structure $*$, namely the functor $(-) * \mathbf{N}$.

Lemma 10.2.7. *Let $\mathbb{B}$ be a boolean topos, let $\mathbf{N}$ be an object of $\mathbb{B}$ and let $*$ be a strict affine symmetric monoidal structure such that $-*\mathbf{N}$ preserves pullbacks. If $(\mathbb{B},*,\mathbf{N})$ satisfies the properties in Propositions 10.2.5 and 10.2.6, then $(\mathbb{B},*,\mathbf{N})$ is a category with binding structure.*

For the proof of this lemma, we use two sub-lemmas.

Lemma 10.2.8. *Under the assumptions of Lemma 10.2.7, the morphism $\eta' \colon \mathbf{N}\times B \to (\mathbf{N} \multimap B)*\mathbf{N}$ is an isomorphism with inverse $\langle \dot{\pi}_2, \varepsilon\rangle \colon (\mathbf{N}\multimap B)*\mathbf{N} \to \mathbf{N}\times B$.*

Proof. The equation $\langle \dot{\pi}_2,\varepsilon\rangle \circ \eta' = id$ holds because the diagram in Proposition 10.2.6 commutes. Menni shows that $\langle \dot{\pi}_2,\varepsilon\rangle$ is a monomorphism [72, Lemma 4.1]. Because of $\langle \dot{\pi}_2,\varepsilon\rangle \circ \eta' = id$, we have $\langle \dot{\pi}_2,\varepsilon\rangle \circ \eta' \circ \langle \dot{\pi}_2,\varepsilon\rangle = \langle \dot{\pi}_2,\varepsilon\rangle$. Since $\langle \dot{\pi}_2,\varepsilon\rangle$ is mono, this implies $\eta' \circ \langle \dot{\pi}_2,\varepsilon\rangle = id$, and thus the assertion. □

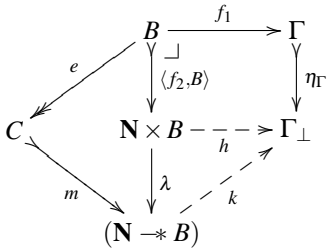
Lemma 10.2.9. *Under the assumptions of Lemma 10.2.7, for each morphism $f \colon B \to \Gamma * \mathbf{N}$, there exists a morphism $g \colon C \to \Gamma$ such that f and $g*\mathbf{N}$ are isomorphic as objects of $\mathbb{B}/(\Gamma*\mathbf{N})$.*

Proof. Let $f \colon B \to \Gamma*\mathbf{N}$ and write f_i as an abbreviation for $\dot{\pi}_i \circ f$ for $i = 1,2$.

Menni shows in [72, Lemma 5.6] that, for each map $h \colon \mathbf{N}\times B \to \Gamma$ for which in the internal logic of $\mathbb{B}$ the sequent $n\colon \mathbf{N}, b\colon B \mid \top \vdash n \# h(n,b)$ holds, there exists a unique map $k \colon (N \multimap B) \to \Gamma$ satisfying $h = k \circ \lambda$. Here, $\lambda : \mathbf{N}\times B \to (\mathbf{N}\multimap B)$ is the map from Proposition 10.2.6.

Let h be the partial map classifier of $(\langle f_2, B\rangle, f_1)$ as in the square in the diagram below. We show that $n\colon \mathbf{N}, b\colon B \mid \top \vdash n \# h(n,b)$ holds. In a boolean topos we can assume $\Gamma_\perp$ to be $\Gamma + 1$, see e.g. [57, Lemma 2.4.6]. Hence we can assume $(\exists \gamma.\, h(n,b) = \kappa_1(\gamma)) \vee (h(n,b) = \kappa_2(\mathrm{unit}))$. We continue by case distinction. In case $h(n,b) = \kappa_1(\gamma)$, we know $\gamma = f_1(b)$ and $n = f_2(b)$ since the square in the diagram below commutes. Since the codomain of f is $\Gamma*\mathbf{N}$, we get $\gamma = f_1(b) \# f_2(b) = n$, which implies the required $n \# \kappa_1(\gamma) = h(n,b)$. In the case where $h(n,b) = \kappa_2(\mathrm{unit})$ holds, $n \# h(n,b)$ is trivial since 1 is the unit of $(-)*\mathbf{N}$, so that $(n \# \mathrm{unit})$ holds for arbitrary names n.

Having shown the property $n\colon \mathbf{N}, b\colon B \mid \top \vdash n \# h(n,b)$, [72, Lemma 5.6] provides us with a unique map k making the diagram below commute.



Let the maps e and m shown in the above diagram be the epi-mono factorisation of $\lambda \circ \langle f_2, B\rangle$. It follows straightforwardly that $k \circ m \colon C \to \Gamma_\perp$ factors through $\eta_\Gamma \colon \Gamma \to \Gamma_\perp$, say via a map $g \colon C \to \Gamma$.

We show that f and $g * \mathbf{N}$ are isomorphic as maps of $\mathbb{B}/\Gamma * \mathbf{N}$. We start by showing the existence of (necessarily unique) maps u and v as in the two diagrams below.

$$\begin{array}{ccc} B & \overset{u}{\dashrightarrow} & C * \mathbf{N} \\ {\scriptstyle \langle f_2, B\rangle}\downarrow & & \downarrow{\scriptstyle m*\mathbf{N}} \\ \mathbf{N} \times B & \xrightarrow[\eta']{} & (\mathbf{N} \multimap * B) * \mathbf{N} \end{array} \qquad \begin{array}{ccc} C * \mathbf{N} & \overset{v}{\dashrightarrow} & B \\ {\scriptstyle m*\mathbf{N}}\downarrow & & \downarrow{\scriptstyle \langle f_2, B\rangle} \\ (\mathbf{N} \multimap * B) * \mathbf{N} & \xrightarrow[\langle \dot\pi_2, \varepsilon\rangle]{} & \mathbf{N} \times B \end{array} \tag{10.2}$$

Note that we assume $(-) * \mathbf{N}$ to preserve pullbacks, so that it must preserve monos. The existence of u follows because we can use an argument as in the proof of Proposition 10.2.4 to show that the following square is a pullback.

$$\begin{array}{ccc} C * \mathbf{N} & \xrightarrow{\dot\pi_1} & C \\ {\scriptstyle m*\mathbf{N}}\downarrow & \lrcorner & \downarrow{\scriptstyle m} \\ (\mathbf{N} \multimap * B) * \mathbf{N} & \xrightarrow[\dot\pi_1]{} & (\mathbf{N} \multimap * B) \end{array}$$

Since $m \circ e = \lambda \circ \langle f_2, B\rangle = \dot\pi_1 \circ \eta' \circ \langle f_2, B\rangle$ we can use the pullback to show that $\eta' \circ \langle f_2, B\rangle$ factors through $m * \mathbf{N}$. This implies the existence of the map u.

For the existence of v, we define $v \overset{\text{def}}{=} \pi_2 \circ \langle \dot\pi_2, \varepsilon\rangle \circ (m * \mathbf{N})$. We have to show that the so defined v makes the square commute. By definition $\pi_2 \circ \langle f_2, B\rangle \circ v = \pi_2 \circ \langle \dot\pi_2, \varepsilon\rangle \circ (m * \mathbf{N})$ holds. It therefore suffices to show $\pi_1 \circ \langle f_2, B\rangle \circ v = \pi_1 \circ \langle \dot\pi_2, \varepsilon\rangle \circ (m * \mathbf{N})$. For this it suffices to show $f_2 \circ \varepsilon \circ (m * \mathbf{N}) = \dot\pi_2$. Expressed in the internal language, this amounts to

$$c\colon (\mathbf{N} \multimap * B), n\colon \mathbf{N} \mid c \# n, \exists b\colon B.\, c = (f_2(b).b) \vdash f_2(c@n) = n. \tag{10.3}$$

In this sequent we write $(-).(-)$ for the term $\lambda(-,-)$. In order to prove this sequent, we first show the following sequent.

$$c\colon (\mathbf{N} \multimap * B) \mid \exists b\colon B.\, c = f_2(b).b \vdash \mathcal{N} n.\, f_2(c@n) = n \tag{10.4}$$

By using the existential rule on the left and the replacement rule for equality, it suffices to show the equation $\mathcal{N} n.\, f_2((f_2(b).b)@n) = n$. Since $f_2(b)$ is fresh for $(f_2(b).b)$, we can use the existential view of $\mathcal{N}$ with $f_2(b)$, and it suffices to show $f_2((f_2(b).b)@f_2(b)) = f_2(b)$. This follows directly from the equation $(n.x)@n = x$, which is available since the diagram in Proposition 10.2.6 commutes by assumption. With (10.4), showing (10.3) is straightforward. In (10.3), we can now assume $\mathcal{N} n.\, f_2(c@n) = n$. Because we have $c \# n$, we can use the universal view of $\mathcal{N}$ and instantiate it with n. We have therefore shown the existence of a map v making the right-hand square in 10.2 commute.

By Lemma 10.2.8, η' and $\langle \dot\pi_2, \varepsilon\rangle$ are mutually inverse, which implies that u and v are mutually inverse. To show that u and v constitute an isomorphism between the objects f and $g * \mathbf{N}$ of $\mathbb{B}/(\Gamma * \mathbf{N})$, it just remains to show $f = (g * \mathbf{N}) \circ u$. Since $*$ is strict affine, it suffices to show $\dot\pi_i \circ f = \dot\pi_i \circ (g * \mathbf{N}) \circ u$ for $i = 1, 2$. This follows straightforwardly from the definitions of u and g. □

Proof of Lemma 10.2.7. We have to show that $W_{\mathbf{N}}\colon \mathbb{B}^{\to} \to \mathbb{B}/(- * \mathbf{N})$, defined by mapping $f\colon A \to \Gamma$ to $f * \mathbf{N}\colon A * \mathbf{N} \to \Gamma * \mathbf{N}$ is an equivalence of the fibrations $cod\colon \mathbb{B}^{\to} \to \mathbb{B}$ and $(- * \mathbf{N})^* cod\colon \mathbb{B}/(- * \mathbf{N}) \to \mathbb{B}$.

By Lemma 2.2.9, it suffices to show that $W_{\mathbf{N}}$ is a fibred functor between these fibration, that $W_{\mathbf{N}}$ is full and faithful, and that it is essentially surjective. That $W_{\mathbf{N}}$ is a fibred functor follows directly from the assumption that $(-)*\mathbf{N}$ preserves pullbacks.

Next we show that $W_{\mathbf{N}}$ is faithful. For this, it is enough to show that $(-)*\mathbf{N}$ is faithful. Since we have assumed the property stated in Proposition 10.2.5, we can show as in the proof of Proposition 10.2.3, that the affine projection $\dot{\pi}_1 : A*\mathbf{N} \to A$ is an epimorphism for all objects A. Suppose now that we have $u,v\colon A \to B$ with $u*\mathbf{N} = v*\mathbf{N}$. By naturality of $\dot{\pi}_1$, this implies $u \circ \dot{\pi}_1 = \dot{\pi}_1 \circ (u*\mathbf{N}) = \dot{\pi}_1 \circ (v*\mathbf{N}) = v \circ \dot{\pi}_1$. Since $\dot{\pi}_1$ is epi, we get $u = v$. Hence, $(-)*\mathbf{N}$ is faithful and therefore so is $W_{\mathbf{N}}$.

To show that $W_{\mathbf{N}}$ is full, let (u,v) be a map in $\mathbb{B}/(-*\mathbf{N})$ between objects that are in the image of $W_{\mathbf{N}}$, that is a commuting square of the following form.

$$\begin{array}{ccc} A*\mathbf{N} & \xrightarrow{u} & B*\mathbf{N} \\ {\scriptstyle f*\mathbf{N}}\downarrow & & \downarrow{\scriptstyle g*\mathbf{N}} \\ \Gamma*\mathbf{N} & \xrightarrow[v*\mathbf{N}]{} & \Delta*\mathbf{N} \end{array} \tag{10.5}$$

We have to find a map $u'\colon A \to B$ such that both $u = u'*\mathbf{N}$ and $g \circ u' = v \circ f$ hold. We use unique choice, available in any topos [56, §4.9], to construct u'. Unique choice allows us to construct a morphism from a functional relation. The functional relation we use to construct u' is given by the proposition

$$\forall x\colon A.\, \exists! y\colon B.\, И n.\, \dot{\pi}_1(u(x,n)) = y, \tag{10.6}$$

which we now show holds. We have to establish the sequent

$$x\colon A \mid \top \vdash \exists! y\colon B.\, И n.\, \dot{\pi}_1(u(x,n)) = y.$$

We make use of Proposition 10.2.5, which gives us that $И$ is both an existential and a universal quantifier. By using the universal nature of $И$, it is trivial to prove $И n.\, \top$. Using the existential elimination on this formula, we obtain a fresh name, i.e. we reduce the goal to

$$x\colon A, m\colon \mathbf{N} \mid x \# m \vdash \exists! y\colon B.\, И n.\, \dot{\pi}_1(u(x,n)) = y.$$

We instantiate y with $\dot{\pi}_1(u(x,m))$. Hence, it remains to show

$$x\colon A, m\colon \mathbf{N} \mid x \# m \vdash И n.\, \dot{\pi}_1(u(x,n)) = \dot{\pi}_1(u(x,m)) \tag{10.7}$$

and

$$x\colon A, m\colon \mathbf{N} \mid x \# m \vdash \forall y\colon B.\, (И n.\, \dot{\pi}_1(u(x,n)) = y) \implies \dot{\pi}_1(u(x,m)) = y. \tag{10.8}$$

For the first sequent, we observe that we have the following derivation in the internal logic.

$$\text{(SUBST)}\ \frac{\dfrac{\text{(AXIOM)}\ \dfrac{}{x\colon A, z\colon B, m\colon \mathbf{N} \mid x \# m,\ z \# m,\ \dot{\pi}_1(u(x,m)) = z \vdash \dot{\pi}_1(u(x,m)) = z}}{\text{(И-}\exists\text{-R)}\ \dfrac{x\colon A, z\colon B, m\colon \mathbf{N} \mid x \# m,\ z \# m,\ \dot{\pi}_1(u(x,m)) = z \vdash И n.\, \dot{\pi}_1(u(x,n)) = z}{}}}{x\colon A, m\colon \mathbf{N} \mid x \# m,\ \dot{\pi}_1(u(x,m)) \# m,\ \dot{\pi}_1(u(x,m)) = \dot{\pi}_1(u(x,m)) \vdash И n.\, \dot{\pi}_1(u(x,n)) = \dot{\pi}_1(u(x,m))}$$

But $\dot{\pi}_1(u(x,m)) = \dot{\pi}_1(u(x,m))$ is trivial and $\dot{\pi}_1(u(x,m)) \# m$ holds because so does $\dot{\pi}_2(u(x,m)) = m$, since the above diagram commutes, and because the codomain of u is $B * \mathbf{N}$. Hence, the sequent (10.7) follows from the conclusion of the derivation. For the sequent (10.8), it suffices to show

$$x\colon A \mid \top \vdash И m.\, И n.\, \dot{\pi}_1(u(x,n)) = \dot{\pi}_1(u(x,m)).$$

By the universal nature of И, it suffices to show

$$x\colon A, m\colon \mathbf{N} \mid x \# m \vdash И n.\, \dot{\pi}_1(u(x,n)) = \dot{\pi}_1(u(x,m)).$$

But this is just sequent (10.7), which we have already shown above. Therefore, we have shown sequent (10.6), which allows us to use unique choice to define a map $u'\colon A \to B$. Finally, that $u = u' * \mathbf{N}$ holds now follows from the definition of u' and because diagram (10.5) commutes. Since $(-) * \mathbf{N}$ is faithful and diagram (10.5) commutes, we also obtain $g \circ u' = v \circ f$. This shows that $W_{\mathbf{N}}$ is full.

It remains to show that $W_{\mathbf{N}}$ is essentially surjective. But this we have already shown in Lemma 10.2.9. □

Lemma 10.2.10. *Let $\mathbb{B}$ be a topos, $\mathbf{N}$ be an object of $\mathbb{B}$ and $*$ be a strict affine symmetric monoidal structure such that $- * \mathbf{N}$ preserves pullbacks. If $(\mathbb{B}, *, \mathbf{N})$ satisfies the properties in Propositions 10.2.5 and 10.2.6, then $[\delta, \iota]\colon \mathbf{N} + (\mathbf{N} * \mathbf{N}) \to \mathbf{N} \times \mathbf{N}$ is an isomorphism.*

Proof. By Lemma 4.3 of [72]. □

In a boolean topos Menni's binders therefore give rise to a category with bindable names.

Proposition 10.2.11. *Let $\mathbb{B}$ be a boolean topos, let $\mathbf{N}$ be an object of $\mathbb{B}$ and let $*$ be a strict affine symmetric monoidal structure such that $- * \mathbf{N}$ preserves pullbacks. If $(\mathbb{B}, *, \mathbf{N})$ satisfies the properties in Propositions 10.2.5 and 10.2.6, then $(\mathbb{B}, *, \mathbf{N})$ is a category with bindable names.*

Proof. By Proposition 10.2.4, using Lemmas 10.2.7 and 10.2.10 □

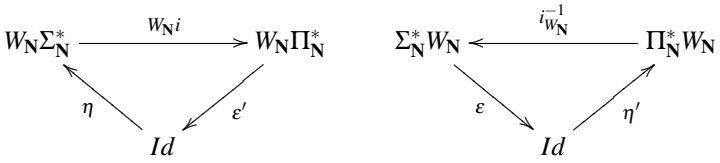
10.3 Instances

In this section we give particular examples of categories with binding structure and categories with bindable names. Most importantly, we show that the categories constructed in Chapter 3 are categories with bindable names.

10.3.1 The Schanuel Topos

The Schanuel topos $\mathbb{S}$ becomes a category with bindable names when we use the monoidal structure for freshness $*$ and the object of names $\mathbf{N}$ as defined in Chapter 3. Although this is already implied by Proposition 10.2.11, we show it directly in terms of the structure defined in Chapter 3. The direct proof

can be generalised to $\mathbf{Ass}_S(P)$. First we describe informally what the binding structure means in the Schanuel topos and how it relates to the work of Gabbay & Pitts. By Proposition 10.1.2, the binding structure $(\mathbb{S}, *, \mathbf{N})$ amounts to a vertical natural isomorphism $i\colon \Sigma^*_{\mathbf{N}} \to \Pi^*_{\mathbf{N}}$ making the triangles below commute.



One way of understanding this situation is as a propositions as types generalisation of the freshness quantifier. Under the propositions as types reading, $\Sigma^*_{\mathbf{N}}$ reads as 'there exists a fresh name' and $\Pi^*_{\mathbf{N}}$ reads as 'for all fresh names'. The freshness quantifier arises because these two propositions are equivalent, so that the isomorphism $\Sigma^*_{\mathbf{N}} \cong \Pi^*_{\mathbf{N}}$ can be understood as a propositions-as-types version of it. Indeed, we have seen in Proposition 10.2.5 that the restriction of $\Sigma^*_{\mathbf{N}}$ and $\Pi^*_{\mathbf{N}}$ to subobjects yields just the freshness quantifier. However, this description does not explain the commuting triangles. Since the subobject fibration is a fibred preorder, commutativity of the triangles is trivial for the freshness quantifier.

A second way of understanding the above situation, which does explain why we want the triangles to commute, is by viewing both $\Sigma^*_{\mathbf{N}}$ and $\Pi^*_{\mathbf{N}}$ as representations of α-equivalence classes. Recall that the elements of type $\Sigma^* n\colon \mathbf{N}.A$ are pairs $n.x$ in which the name n is hidden. The elements of type $\Pi^* n\colon \mathbf{N}.A$ are partial functions defined just on fresh names. As observed in [38], both pairs with hidden names and partial functions defined just on fresh names represent α-equivalence classes. The types $\Sigma^* n\colon \mathbf{N}.A$ and $\Pi^* n\colon \mathbf{N}.A$ should therefore be isomorphic. But this is only part of the story. We also have the introduction and elimination terms for both types, and it is natural to ask what happens if, for example, we first introduce an element of $\Sigma^*_{\mathbf{N}}$, use the isomorphism, and then use the elimination of $\Pi^*_{\mathbf{N}}$. Explaining this is the purpose of the two commuting triangles above. Using a somewhat sloppy notation in which we omit the terms for freefrom types wherever possible, and in which we assume i to be the identity (which we can always do), we explain the triangles informally. The equations $\varepsilon' \circ W_{\mathbf{N}}i \circ \eta = id$ and $W_{\mathbf{N}}i \circ \eta \circ \varepsilon' = id$ of the left triangle can be read as $n.(x@n) = x$ and $(m.y)@m = y$, where x is a function in $\Pi^* n\colon \mathbf{N}.A$, m is an arbitrary name, n is some/any name fresh for x, and y is an element of A. The left triangle thus corresponds to forms of β and η-conversion for the mixed use of (Σ^*-I) and (Π^*-E). These equations appear in [38]. Notice that the equation $n.(x@n) = x$ implies that each function in $\Pi^* n\colon \mathbf{N}.A$ is uniquely determined by its value at a single fresh name. The right-hand triangle expresses that binding a name n in some x for which n is already fresh results in a constant function in $\Pi^*_{\mathbf{N}}$. The syntactic equivalent of $\Sigma^*_{\mathbf{N}}W_{\mathbf{N}}A$ in the left corner of the right-hand triangle is $\Sigma^* n\colon \mathbf{N}.\left(A^{*(n:\mathbf{N})}\right)$. Its elements correspond to pairs-with-hiding $n.x$ in which the name n is fresh for x. The counit ε maps $n.x$ to x, which makes sense only because n is fresh for x. The unit η' maps x to the constant function $\lambda^* n\colon \mathbf{N}.x$. The triangle thus states that, whenever n is fresh for x, the pair $n.x$ is effectively the same as the constant function $\lambda^* n\colon \mathbf{N}.x$.

Proposition 10.3.1. *The Schanuel topos $(\mathbb{S}, *, \mathbf{N})$ is a category with bindable names.*

Proof. We use Proposition 10.2.4. The properties $2'$ and 3 have already been shown in Chapter 3. It remains to show that $(\mathbb{S}, *, \mathbf{N})$ is a category with binding structure. In Chapter 3 we have constructed simple monoidal sums $\Sigma^*_{\mathbf{N}}$ and simple monoidal products $\Pi^*_{\mathbf{N}}$. By Proposition 10.1.2, it therefore suffices to exhibit a vertical isomorphism that makes the two triangles there commute. We will show this for the split fibration from Section 3.5.2, which is equivalent to the codomain fibration. We use the notation from Section 3.5.2 of Chapter 3.

First we define a natural isomorphism $j\colon [\mathbf{N}](-) \to (\mathbf{N} \multimap -)$. Using the internal language of $\mathbb{S}$, we define $j_B([n,x])$ to be $\lambda^* m\colon \mathbf{N}.\, (m\ n)\cdot x$. That this definition is independent of the choice of representative follows from Gabbay & Pitts' characterisation of the elements of $[\mathbf{N}]B$, reproduced here in equation (3.7). The inverse of j_B maps $f \in (\mathbf{N} \multimap B)$ to $[n, f@n]$ where n is some fresh name. It is straightforward to show that this defines an isomorphism, by using the fact that the functions in $(\mathbf{N} \multimap B)$ are uniquely determined by their value at a single fresh name, and by using the characterisation of $[\mathbf{N}]B$ from Lemma 3.3.28. For naturality, let $g\colon B \to C$ be a map in $\mathbb{S}$. Then $j_C \circ ([\mathbf{N}]g)$ maps $[n,x]_{[\mathbf{N}]B}$ to $\lambda^* m\colon \mathbf{N}.\, (m\ n)\cdot g(x)$, while $(\mathbf{N} \multimap g) \circ j_B$ maps $[n,x]_{[\mathbf{N}]B}$ to $\lambda^* m\colon \mathbf{N}.\, g((m\ n)\cdot x)$. By equivariance of g both maps are equal.

Next, we recall the definitions of the objects $\Sigma^*_{\mathbf{N}}(\varphi)\colon \Gamma \times [\mathbf{N}]B \to \Omega$ and $\Pi^*_{\mathbf{N}}(\varphi)\colon \Gamma \times (\mathbf{N} \multimap B) \to \Omega$ for an object $\varphi\colon (\Gamma * \mathbf{N}) \times B \to \Omega$ over $\Gamma * \mathbf{N}$.

$$\Sigma^*_{\mathbf{N}}(\varphi)(\gamma, c) \overset{\text{def}}{\iff} \forall n \in \mathbf{N}.\, \forall b \in B.\, (n \# \langle \gamma, c\rangle \wedge \langle n, b\rangle \in c) \implies \varphi(\gamma, n, b)$$
$$\Pi^*_{\mathbf{N}}(\varphi)(\gamma, f) \overset{\text{def}}{\iff} \forall n \in \mathbf{N}.\, n \# \langle \gamma, f\rangle \implies \varphi(\gamma, n, f@n)$$

Using the definition of the isomorphism j, we obtain an equivalence

$$\Sigma^*_A(\varphi)(\gamma, c) \iff \Pi^*_A(\varphi)(\gamma, j_B(c)).$$

Hence, we get a vertical isomorphism i_φ of the objects $\Sigma^*_{\mathbf{N}}(\varphi)$ and $\Pi^*_{\mathbf{N}}(\varphi)$ as in the following diagram.

$$\begin{array}{ccc}
\{\Sigma^*_{\mathbf{N}}(\varphi)\} & \overset{i_\varphi}{\dashrightarrow} & \{\Pi^*_{\mathbf{N}}(\varphi)\} \\
\downarrow{\scriptstyle m_{\Sigma^*_{\mathbf{N}}(\varphi)}} & & \downarrow{\scriptstyle m_{\Pi^*_{\mathbf{N}}(\varphi)}} \\
\Gamma \times [\mathbf{N}]B & \xrightarrow[\Gamma \times j_B]{} & \Gamma \times (\mathbf{N} \multimap B) \\
\downarrow{\scriptstyle \pi_1} & & \downarrow{\scriptstyle \pi_1} \\
\Gamma & \xrightarrow[id]{} & \Gamma
\end{array}$$

That this defines a natural isomorphism $i\colon \Sigma^*_{\mathbf{N}} \to \Pi^*_{\mathbf{N}}$ follows analogously to the naturality of j.

We come to verifying that the two triangles from Proposition 10.1.2 commute. First we show that the following triangle in $\mathbb{S}/(\Gamma * \mathbf{N})$ commutes.

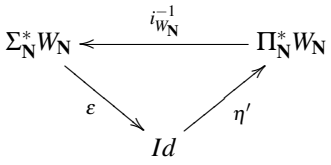
$$\begin{array}{ccc}
W_{\mathbf{N}}\Sigma^*_{\mathbf{N}} & \xrightarrow{W_{\mathbf{N}} i} & W_{\mathbf{N}}\Pi^*_{\mathbf{N}} \\
& {\scriptstyle \eta} \nwarrow \quad \swarrow {\scriptstyle \varepsilon'} & \\
& Id &
\end{array}$$

It suffices to check $W_{\mathbf{N}}i \circ \eta \circ \varepsilon' = id$ and $\varepsilon' \circ W_{\mathbf{N}}i \circ \eta = id$. After expanding the definitions, these equations amount to the equations $n.(f@n) = f$ and $(n.x)@n = x$, known from [38]. We spell out the details. For the first equation, let $\langle \gamma, n, f\rangle \in \{W_{\mathbf{N}}\Pi^*_{\mathbf{N}}(\varphi)\}$. We have $\varepsilon'(\langle \gamma, n, f\rangle) = \langle \gamma, n, f@n\rangle$, $\eta(\langle \gamma, n, f@n\rangle) = \langle \gamma, n, [n, f@n]_{[\mathbf{N}]B}\rangle$ and $W_{\mathbf{N}}i_{\varphi}(\langle \gamma, n, [n, f@n]_{[\mathbf{N}]B}\rangle) = \langle \gamma, n, \lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot(f@n)\rangle$. Since $\langle \gamma, n, f\rangle \in \{W_{\mathbf{N}}\Pi^*_{\mathbf{N}}(\varphi)\}$, the definition of $W_{\mathbf{N}}$ implies that n is fresh for f. Hence, $(m\ n)\cdot f = f$ holds for any fresh name m. Therefore, we can use equivariance and η-equality for the monoidal exponent to get the following equations:

$$\begin{aligned}\lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot(f@n) &= \lambda^* m\colon \mathbf{N}.\,((m\ n)\cdot f)@((m\ n)\cdot n)\\ &= \lambda^* m\colon \mathbf{N}.\, f@m\\ &= f\end{aligned}$$

This shows the first equation. For the second equation, let $\langle \gamma, n, x\rangle \in \{\varphi\}$. Then we have $\eta(\langle \gamma, n, x\rangle) = \langle \gamma, n, [n, x]_{[\mathbf{N}]B}\rangle$ and $W_{\mathbf{N}}i_{\varphi}(\langle \gamma, n, [n, x]_{[\mathbf{N}]B}\rangle) = \langle \gamma, n, \lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot x\rangle$ and $\varepsilon'(\langle \gamma, n, \lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot x\rangle) = \langle \gamma, n, (\lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot x)@n\rangle$. But we have $(\lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot x)@n = (n\ n)\cdot x = x$, which shows the second equation.

To show that the triangle in $\mathbb{S}/\Gamma$



commutes, it suffices to check $i_{W_{\mathbf{N}}} = \eta' \circ \varepsilon$ and $\varepsilon \circ i^{-1}_{W_{\mathbf{N}}} \circ \eta' = id$. For the first equation, let $\langle \gamma, c\rangle \in \{\Sigma^*_{\mathbf{N}}W_{\mathbf{N}}\psi\}$. Then we have $\varepsilon(\langle \gamma, c\rangle) = \langle \gamma, b\rangle$ where $c = [n, b]_{[\mathbf{N}]B}$ such that n is fresh for γ. Notice that n must also be fresh for b since $\langle \gamma, c\rangle$ is in $\{\Sigma^*_{\mathbf{N}}W_{\mathbf{N}}\psi\}$. Furthermore, $\eta'(\langle \gamma, b\rangle) = \langle \gamma, \lambda^* m\colon \mathbf{N}.\, b\rangle$ and $i_{W_{\mathbf{N}}}(\langle \gamma, c\rangle) = \langle \gamma, \lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot b\rangle$. Since n is fresh for b, we have $\lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot b = \lambda^* m\colon \mathbf{N}.\, b$, which implies the first equation. For the second equation, let $\langle \gamma, x\rangle \in \{\psi\}$. Then, $\eta'(\langle \gamma, x\rangle) = \langle \gamma, \lambda^* n\colon \mathbf{N}.\, x\rangle$ and $i^{-1}_{W_{\mathbf{N}}}(\langle \gamma, \lambda^* n\colon \mathbf{N}.\, x\rangle) = \langle \gamma, [m, (\lambda^* n\colon \mathbf{N}.\, x)@m]_{[\mathbf{N}]B}\rangle$ where m is an arbitrary fresh name. By β-reduction, we obtain $i^{-1}_{W_{\mathbf{N}}}(\langle \gamma, \lambda^* n\colon \mathbf{N}.\, x\rangle) = \langle \gamma, [m, x]_{[\mathbf{N}]B}\rangle$. Finally, we have $\varepsilon(\langle \gamma, [m, x]_{[\mathbf{N}]B}\rangle) = \langle \gamma, x\rangle$, as required. □

It is worth summarising the structure we get from this proposition in relation to the structure constructed in [38]. The functor $\mathbf{N} \multimap (-)$ corresponds to the abstraction set of [38]. To work with $\mathbf{N} \multimap (-)$ we can use the units and counits of both $\Sigma^*_{\mathbf{N}}$ and $\Pi^*_{\mathbf{N}}$, which corresponds to the dual view of abstractions sets in [38] as pairs with name-hiding and as partial functions. How these two views interact is explained by the two commuting triangles, which make available equations similar to those in [38]. Furthermore, we have also seen that the binding structure of the Schanuel topos implies the existence of the freshness quantifier И. Finally, Propositions 10.1.5 and 10.1.6 show that our axiomatisation implies the remarkable properties of И and $\mathbf{N} \multimap (-)$ that have been shown in [38].

The binding structure $(\mathbb{S}, *, \mathbf{N})$ is not the only binding structure in the Schanuel topos. We can generalise the above proposition from names $\mathbf{N}$ to name-like objects. In the rest of this section, we define name-like objects as the objects with essentially simple transitive action and we show that for such objects A the triple $(\mathbb{S}, *, A)$ is a binding structure.

Definition 10.3.2. An object A of $\mathbb{S}$ has *transitive action* if, for any two elements x and y of A, there exists a permutation π such that $\pi \cdot x = y$ holds.

Lemma 10.3.3. *Let A be an object of $\mathbb{S}$ with transitive action. For any two elements x and y of A, there exists a permutation π that satisfies $\pi \cdot x = y$ and $\pi(n) = n$ for all $n \notin \mathrm{supp}(x) \cup \mathrm{supp}(y)$.*

For the rest of this section, we choose, for any two elements x and y of A, a permutation $\pi_{x,y}$ that satisfies $\pi_{x,y} \cdot x = y$ and $\pi_{x,y}(n) = n$ for all $n \notin \mathrm{supp}(x) \cup \mathrm{supp}(y)$. We make this choice for convenience: It simplifies the presentation, as we do not have to introduce the permutations $\pi_{x,y}$ explicitly. We remark, however, that we make the choice for convenience only, and that the development in the rest of this section can be carried out without it.

Definition 10.3.4. An object A of $\mathbb{S}$ has an *essentially simple action* if, for all $x \in A$, any two permutations π and τ that satisfy $\pi \cdot x = \tau \cdot x$ also satisfy $\forall n \in \mathrm{supp}(x).\, \pi(n) = \tau(n)$.

The objects with essentially simple transitive action have a simple characterisation. Define inductively $\mathbf{N}^{*0} = 1$ and $\mathbf{N}^{*k} = \mathbf{N} * \mathbf{N}^{*(k-1)}$ for any $k \geq 1$.

Lemma 10.3.5. *For each $k \geq 0$, the object $\mathbf{N}^{*k}$ has essentially simple transitive action.*

Lemma 10.3.6. *Any non-empty set A with essentially simple transitive action is isomorphic to $\mathbf{N}^{*k}$ for some $k \geq 0$.*

Proof. Let A be a non-empty set with essentially simple transitive action. Because the action on A is transitive, the support of all its elements must contain the same (finite) number of names. Let k be this number, i.e. $k \geq 0$ is such that $\forall x \in A.\, k = |\mathrm{supp}(x)|$ holds.

We define a map $i\colon A \to \mathbf{N}^{*k}$. Let x be an arbitrary element of A. Let $n_1, \ldots, n_k$ be the names in $\mathrm{supp}(x)$ in some arbitrary order. Define $i(\pi \cdot x) \stackrel{\mathrm{def}}{=} \langle \pi(n_1), \ldots, \pi(n_k) \rangle$, where π is an arbitrary permutation. This assignment is functional, since from the assumption that the action on A is essentially simple, we obtain that $\pi \cdot x = \tau \cdot x$ implies that $\pi(n) = \tau(n)$ holds for all $n \in \mathrm{supp}(x)$, and thus $i(\pi \cdot x) = i(\tau \cdot x)$. Furthermore, $i(y)$ is defined for all $y \in A$, since the assumption that the action on A is transitive yields that, for all $y \in A$, there exists a permutation π satisfying $\pi \cdot x = y$. Finally, the assignment is equivariant by definition. Hence, the assignment defines a morphism $i\colon A \to \mathbf{N}^{*k}$ in $\mathbb{S}$.

An inverse to i can be defined by mapping $\langle m_1, \ldots, m_k \rangle \in \mathbf{N}^{*k}$ to $\pi \cdot x \in A$, where π is an arbitrary permutation satisfying $\pi(n_i) = m_i$ for all $i \in \{1, \ldots, k\}$. This assignment is functional since π is completely determined on $\mathrm{supp}(x)$. Since equivariance is straightforward to show, we have thus defined a morphism $i^{-1}\colon \mathbf{N}^{*k} \to A$ in $\mathbb{S}$. Checking that i^{-1} is inverse to i is routine. □

We remark that in the functor category presentation of $\mathbb{S}$, the objects of the form $\mathbf{N}^{*k}$ are precisely the representable sheaves.

Proposition 10.3.7. *The triple $(\mathbb{S}, *, A)$ is a category with binding structure whenever A is a non-empty set with essentially simple transitive action.*

In the proof we use the following three lemmas.

Lemma 10.3.8. *Let A be an object of $\mathbb{S}$ with essentially simple transitive action. For all $f, g \in (A \multimap B)$, if there exists $x \in A$ such that both $\langle f, g\rangle \# x$ and $f@x = g@x$ hold, then $f = g$ holds.*

Proof. Let $x \in A$ with $\langle f, g\rangle \# x$ and suppose $f@x = g@x$. For any $y \in A$ with $\langle f, g\rangle \# y$, we have $f@y = f@\pi_{x,y} \cdot x = \pi_{x,y} \cdot (f@x)$. The last step holds because both x and y are fresh for f, and so $\pi_{x,y} \cdot f = f$ holds. Likewise, we get $g@y = \pi_{x,y} \cdot (g@x)$. Since using $f@x = g@x$, this implies $f@y = g@y$, we have therefore shown that f and g agree on all arguments y that are fresh for them both. By definition of $(A \multimap B)$ as $E/\sim$ in Section 3.3.1, f and g therefore denote the same element of $(A \multimap B)$. □

Next we consider the structure of the set $[A]B$, as defined in Section 3.5.2. Recall that $[A]B$ is defined as a quotient of $A \times B$ with respect to an equivalence relation $\sim_{[A]B}$. The following lemmas are special cases of the results in Section 3.3.2 for objects with essentially simple transitive action.

Lemma 10.3.9. *If A has essentially simple transitive action then the following equivalence holds for all $x, y \in A$ and all $b, b' \in B$.*

$$\langle x, b\rangle \sim_{[A]B} \langle y, b'\rangle \iff \mathrm{supp}(y) \cap (\mathrm{supp}(b) \setminus \mathrm{supp}(x)) = \emptyset \ \wedge \ \pi_{x,y} \cdot b = b'$$

Proof. By Lemma 3.3.28, we have

$$\langle x, b\rangle \sim_{[A]B} \langle y, b'\rangle \iff \exists \pi.\ (\forall n \in \mathrm{supp}(\langle x, b\rangle) \setminus \mathrm{supp}(x).\ \pi(n) = n) \ \wedge \ \pi \cdot \langle x, b\rangle = \langle y, b'\rangle.$$

It therefore suffices to show that the right-hand sides of the two equivalences imply each other. That $\mathrm{supp}(y) \cap (\mathrm{supp}(b) \setminus \mathrm{supp}(x)) = \emptyset \ \wedge \ \pi_{x,y} \cdot b = b'$ implies $\exists \pi.\ (\forall n \in \mathrm{supp}(\langle x, b\rangle) \setminus \mathrm{supp}(x).\ \pi(n) = n) \ \wedge \ \pi \cdot \langle x, b\rangle = \langle y, b'\rangle$ is straightforward, since we can use $\pi_{x,y}$ for π. In the other direction, we observe that $\mathrm{supp}(y) \cap (\mathrm{supp}(b) \setminus \mathrm{supp}(x)) = \emptyset$ holds, since $\pi \cdot x = y$ implies $\mathrm{supp}(y) = \pi(\mathrm{supp}(x))$ and π is the identity on $\mathrm{supp}(\langle x, b\rangle) \setminus \mathrm{supp}(x)$. It follows from the fact that the action on A is essentially simple that π and $\pi_{x,y}$ agree on $\mathrm{supp}(\langle x, b\rangle)$. This gives $\pi_{x,y} \cdot b = b'$, thus completing the proof. □

Lemma 10.3.10. *Let A be an object of $\mathbb{S}$ with simple transitive action, and let B be an arbitrary object of $\mathbb{S}$. Then any element $c \in [A]B$ is the graph of a partial function $f \in (A \Rightarrow B_\perp)$ with the property $\forall x \in A.\, f \# x \iff f(x) \neq \perp$. Moreover, for all $\langle x, b\rangle \in c$ and $y \in A$, $f \# y$ implies $f(y) = \pi_{x,y} \cdot b$.*

Proof. Let $c \in [A]B$. To see that c is a functional relation, suppose $\langle x, b\rangle \in c$ and $\langle x, b'\rangle \in c$. Since c is an equivalence class of $\sim_{[A]B}$, this implies $\langle x, b\rangle \sim_{[A]B} \langle x, b'\rangle$. By Lemma 10.3.9, this implies $\pi_{x,x} \cdot b = b'$. Since $\pi_{x,x}$ must be the identity, this gives $b = b'$, thus showing that c is the graph of a partial function.

Let $f \in (A \Rightarrow B_\perp)$ be the partial function whose graph is c. To show the equivalence asserted in the lemma let $x \in A$. From left to right, assume $f \# x$. We have to show that there exists $b \in B$ such that $\langle x, b\rangle \in c$. By definition of $[A]B$, we know that c is not empty, say $c = [y, b']_{\sim_{[A]B}}$. Since $\mathrm{supp}(c) = \mathrm{supp}(b') \setminus \mathrm{supp}(y)$ by Corollary 3.3.30 and since $\mathrm{supp}(f) = \mathrm{supp}(c)$, we know that both x and y are fresh for c. Hence, $\pi_{y,x} \cdot c = c$. However, the permutation action is defined pointwise on equivalence classes, so that we have $c = \pi_{y,x} \cdot [y, b']_{\sim_{[A]B}} = [\pi_{y,x} \cdot y, \pi_{y,x} \cdot b']_{\sim_{[A]B}} = [x, \pi_{y,x} \cdot b']_{\sim_{[A]B}}$. Hence, $\langle x, \pi_{y,x} \cdot b'\rangle \in c$, showing the required $f(x) \neq \perp$. For the direction from right to left, assume $f(x) \neq \perp$, i.e. there exists a unique $b \in B$ such that $\langle x, b\rangle \in c$. We have to show $f \# x$. But this follows from $\mathrm{supp}(f) = \mathrm{supp}(c) = \mathrm{supp}(b) \setminus \mathrm{supp}(x)$. □

Proof of Proposition 10.3.7. The proof goes analogously to that of Proposition 10.3.1. This time, we have to find an isomorphism $j\colon [A](-) \to (A \multimap -)$. Like in Proposition 10.3.1, it is possible to define it using internal language of $\mathbb{S}$. To complement the use of the internal language there, we spell out the isomorphism explicitly here. Recall the definition of $A \multimap B$:

$$\begin{aligned} E &\stackrel{\text{def}}{=} \{f \in (A \Rightarrow B_\perp) \mid \forall a \in A.\, a \# f \iff f(a) \neq \perp\} \\ {\sim} &\stackrel{\text{def}}{=} \{\langle f, g\rangle \in E \times E \mid \forall a \in A.\, a \# \langle f, g\rangle \implies f(a) = g(a)\} \\ (A \multimap B) &\stackrel{\text{def}}{=} E/{\sim} \end{aligned}$$

By Lemma 10.3.10, any element $c \in [A]B$ is the graph of a partial function in E.

We define $j_B\colon [A]B \to (A \multimap B)$ to be the function mapping $c \in [A]B$ to the equivalence class under $\sim$ of the partial function in E whose graph is c. We have shown in Lemma 10.3.10 that each $c \in [A]B$ is indeed the graph of a partial function. We show that j_B is an isomorphism. Its inverse j_B^{-1} is given by mapping $[f]_\sim \in (A \multimap B)$ to $[x, f(x)]_{[A]B}$, where $x \in A$ is an arbitrary fresh value. Note that such an $x \in A$ exists since A is non-empty. Moreover, $f(x) \neq \perp$ holds because of $f \in E$. Independence of the choice of representative can be seen as follows. Suppose $[f]_\sim = [g]_\sim$ and let $x \in A$ be fresh for f and $y \in A$ be fresh for g. We have to show $[x, f(x)]_{[A]B} = [y, g(y)]_{[A]B}$. Let $z \in A$ be fresh for f and g.

$$\begin{aligned} [x, f(x)]_{[A]B} &= [z, \pi_{x,z} \cdot (f(x))]_{[A]B} && \text{by Lemma 10.3.9} \\ &= [z, f(\pi_{x,z} \cdot x)]_{[A]B} && \text{since both } x \text{ and } z \text{ are fresh for } f \\ &= [z, f(z)]_{[A]B} && \text{by definition of } \pi_{x,z} \\ &= [z, g(z)]_{[A]B} && \text{since } f(z) = g(z) \text{ by } f \sim g \\ &= [y, g(y)]_{[A]B} && \text{as for } f \end{aligned}$$

This shows that j_B^{-1} is well-defined.

Given $[x, b] \in [A]B$, applying first j_B and then j_B^{-1} gives $[y, \pi_{x,y} \cdot b]$ for some y with $\mathrm{supp}(y) \cap (\mathrm{supp}(b) \setminus \mathrm{supp}(x)) = \emptyset$. This is the case because $\mathrm{supp}([x, b]_{\sim_{[A]B}}) = (\mathrm{supp}(b) \setminus \mathrm{supp}(x))$ holds by Corollary 3.3.30, and $j_B([x, b]_{\sim_{[A]B}})(y) = \pi_{x,y} \cdot b$ holds by Lemma 10.3.10. Since both x and y are fresh for $[x, b]_{\sim_{[A]B}}$, we have $\pi_{x,y} \cdot [x, b]_{\sim_{[A]B}} = [x, b]_{\sim_{[A]B}}$. By definition of the action this gives $\pi_{x,y} \cdot [x, b]_{\sim_{[A]B}} =$

$[\pi_{x,y}\cdot x, \pi_{x,y}\cdot b]_{\sim_{[A]B}} = [y, \pi_{x,y}\cdot b]_{\sim_{[A]B}}$. Hence, we have $j_B^{-1}\circ j_B = id$. Given $[f]_\sim \in (A \multimap B)$, applying first j_B^{-1} and then j_B gives $[g]_\sim \in (A \multimap B)$ satisfying $g(y) = \pi_{x,y}\cdot f(x)$, where x and y are some fresh elements of A. Since the elements of $(A \multimap B)$ are uniquely determined by their value at a single fresh argument (Lemma 10.3.8), for $j_B \circ j_B^{-1} = id$ it suffices to show $f(y) = \pi_{x,y}\cdot f(x)$. But this is immediate since we have $\pi_{x,y}\cdot f = f$ using freshness.

The rest of the verifications are just as in Proposition 10.3.1. □

For the empty set 0, which trivially has essentially simple transitive action, the proposition is not true. This can be seen by observing that $[0]1$ is empty while $(0 \multimap 1)$ contains one element.

Whilst any non-empty object A with essentially simple transitive action makes $(\mathbb{S}, *, A)$ a category with binding structure, the triple $(\mathbb{S}, *, A)$ is a category with bindable names only if A is isomorphic to $\mathbf{N}$. If $A \cong 1$, we have $A * A \cong A \times A$, so that $\iota\colon A * A \rightarrowtail A \times A$ cannot be the complement of the diagonal. If $A \cong N^{*k}$ for $k > 1$, then $\iota\colon A * A \rightarrowtail A \times A$ is also not the complement of the diagonal, since, for $m \neq n$, the two elements $\langle m, n_1, \ldots, n_k\rangle$ and $\langle r, n_1, \ldots, n_k\rangle$ of $\mathbf{N}^{*k}$ are different but not fresh for each other, i.e. not in $\mathbf{N}^{*k} * \mathbf{N}^{*k}$.

It may be possible to lift the assumption that the action on A be essentially simple. However, if we just assume that A has transitive action, then the elements of $[A]B$ are not always functional relations. For example the action on the object $\mathscr{P}_2(\mathbf{N})$, consisting of sets of exactly two names, is transitive but not essentially simple. The equivalence class $[\{m,n\}, n]$ in $[\mathscr{P}_2(\mathbf{N})]\mathbf{N}$ contains both $\langle\{m,n\}, n\rangle$ and $\langle\{m,n\}, m\rangle$, and so is not a functional relation. By making choices (of names), it may be possible to generalise the above propositions to objects like $\mathscr{P}_2(\mathbf{N})$. We leave the details for further work.

It may be the case that the binding structures $(\mathbb{S}, *, \mathbf{N}^{*k})$ follow from the binding structure $(\mathbb{S}, *, \mathbf{N})$ by use of isomorphisms such as $\Pi^* m\colon \mathbf{N}.\Pi^* n\colon \mathbf{N}.B(m,n) \cong \Pi^* p\colon \mathbf{N} * \mathbf{N}.B(\dot\pi_1(p), \dot\pi_2(p))$. It may be simpler to obtain the binding structures $(\mathbb{S}, *, \mathbf{N}^{*k})$ in this way, but the details remain to be worked out.

We end this section by observing that, as a consequence of Proposition 10.3.7, the characterisation of the monoidal closed structure $(A \multimap B)$ from Section 3.3.1 can be simplified if A is an object with essentially simple transitive action.

Proposition 10.3.11. *If A is an object of $\mathbb{S}$ with essentially simple transitive action and B is an arbitrary object of $\mathbb{S}$, then we have the following isomorphism.*

$$(A \multimap B) \cong \{f \in (A \Rightarrow B_\perp) \mid \forall x \in A.\quad (f \# x \iff f(x) \neq \perp) \\ \wedge (f \# x \implies \mathrm{supp}(f) = \mathrm{supp}(f(x)) \setminus \mathrm{supp}(x))\}$$

Proof. The case where A is the empty set is trivial. Assume A non-empty. Consider the monomorphism $m\colon (A \multimap B) \to (A \Rightarrow B_\perp)$ from Proposition 3.3.15. First we show that if $f = m(c)$ then

$$\forall x \in A.\quad (f \# x \iff f(x) \neq \perp) \\ \wedge (f \# x \implies \mathrm{supp}(f) = \mathrm{supp}(f(x)) \setminus \mathrm{supp}(x)). \tag{10.9}$$

Let $x \in A$. That $(m(c) \# x \iff m(c)(x) \neq \perp)$ holds follows from Lemma 3.3.19. For the implication $(m(c) \# x \implies \mathrm{supp}(m(c)) = \mathrm{supp}(m(c)(x)) \setminus \mathrm{supp}(x))$ suppose $m(c) \# x$. Hence, we have $m(c)(x) \neq \perp$.

By definition of m in Proposition 3.3.15, there exists $g \in c$ such that $g(x) = m(c)(x)$. By definition of the equivalence class c, this also implies $g \# x$. By construction of the isomorphism j in the proof of Proposition 10.3.7 above, we have $\langle x, g(x)\rangle \in j^{-1}(c)$. By Corollary 3.3.30, we get $\mathrm{supp}(j^{-1}(c)) = \mathrm{supp}(\langle x, g(x)\rangle) \setminus \mathrm{supp}(x)$. Since monomorphisms leave the support of all elements unchanged, we have $\mathrm{supp}(m(c)) = \mathrm{supp}(c) = \mathrm{supp}(j^{-1}(c))$. Thus, we have $\mathrm{supp}(m(c)) = \mathrm{supp}(g(x)) \setminus \mathrm{supp}(x)$. Since $m(c)(x) = g(x)$, this implies $\mathrm{supp}(m(c)) = \mathrm{supp}(m(c)(x)) \setminus \mathrm{supp}(x)$, thus showing that $m(c)$ satisfies the required formula.

On the other hand, suppose $f \in (A \Rightarrow B_\perp)$ satisfies (10.9). We show that $f = m([f]_\sim)$ holds, where $\sim$ is the equivalence class used in the definition of $(A \multimap B)$. By Lemma 3.3.18, $m([f]_\sim)$ has minimal support among all the elements of $[f]_\sim$. Moreover, as shown in Proposition 3.3.20, there can be at most one element of $[f]_\sim$ with minimal support. Therefore, to show $f = m([f]_\sim)$ it suffices to show $\mathrm{supp}(f) = \mathrm{supp}(m([f]_\sim))$. Since both f and $m([f]_\sim)$ satisfy (10.9), we have $\mathrm{supp}(f) = \mathrm{supp}(f(x)) \setminus \mathrm{supp}(x)$ and $\mathrm{supp}(m([f]_\sim)) = \mathrm{supp}(m([f]_\sim)(x)) \setminus \mathrm{supp}(x)$ for all $x \in A$ that are fresh for both f and $[f]_\sim$. Since A is non-empty, such an x exists. By definition of m, $m([f]_\sim)(x) = f(x)$ holds for such x. Using this we can conclude $\mathrm{supp}(f) = \mathrm{supp}(m([f]_\sim))$, and thus $f = m([f]_\sim)$.

In conclusion, we have shown that the set of partial functions satisfying (10.9) is isomorphic to the image of $m\colon (A \multimap B) \rightarrowtail (A \Rightarrow B_\perp)$, and this shows the claimed isomorphism. □

10.3.2 The Realizability Category $\mathbf{Ass}_S(P)$

Proposition 10.3.12. *The realizability category* $(\mathbf{Ass}_S(P), *, \mathbf{N})$ *is a category with bindable names.*

Proof. To show that $(\mathbf{Ass}_S(P), *, \mathbf{N})$ is a category with binding structure, we check that the isomorphism $j_B\colon [\mathbf{N}]B \to (\mathbf{N} \multimap B)$ in the proof of Proposition 10.3.1 is an isomorphism in $\mathbf{Ass}_S(P)$, i.e. that j and its inverse are realizable.

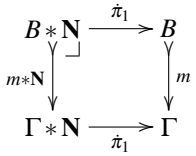
For realizability of j, recall that $j([n,x])$ is defined to be $\lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot x$. Since the realizers of an equivalence class are defined by $||[n,x]|| = \bigcup_{\langle m,y\rangle \in [n,x]} ||\langle m,y\rangle||$, it suffices to realize the mapping $\langle n,x\rangle \mapsto \lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot x$. But given $\langle n,x\rangle$, there exists a realizer r of $\lambda m\colon \mathbf{N}.\,(m\ n)\cdot x$, since the permutation action is realizable. It then follows that r realizes $\lambda^* m\colon \mathbf{N}.\,(m\ n)\cdot x$ too.

For realizability of j^{-1}, recall that $j^{-1}([f]_\sim)$ is defined to be $[n, f(n)]$ where n is a name fresh for f (and therefore for $[f]_\sim$). Again by the pointwise definition of the realizers of equivalence classes, it suffices to realize the mapping $f \mapsto \langle n, f(n)\rangle$. But this is straightforward, since using support approximations we can always generate a fresh name n.

For the binding structure it just remains to check that the two triangles from Proposition 10.1.2 commute. Since this is a property of the underlying sets, it follows just as in 10.3.1.

Since $\mathbf{Ass}_S(P)$, as any quasi-topos, is a coherent category, it just remains to verify points 2 and 3 in Definition 10.2.1. For point 2, we have to show that, for each monomorphism $m\colon B \rightarrowtail \Gamma$, the following

square is a pullback.



We know this to be true in $\mathbb{S}$. Therefore, for any two maps $f\colon C \to \Gamma * \mathbf{N}$ and $g\colon C \to B$ in $\mathbf{Ass}_S(P)$ with $\dot{\pi}_1 \circ f = m \circ g$, there exists a unique map of the underlying sets $u\colon |C| \to |B * \mathbf{N}| = |B| * |\mathbf{N}|$ such that $|f| = |m * \mathbf{N}| \circ |u|$ and $|g| = |\dot{\pi}_1| \circ |u|$. Since the identity of morphisms in $\mathbf{Ass}_S(P)$ is completely determined by the identity of their behaviour on the underlying sets, it suffices to show that u is realized. But if r and s are realizers of f and g respectively, then $\lambda x.\mathit{pair}\ (\mathit{fst}\ (s\ x))\ (\mathit{snd}\ (r\ x))$ is a realizer of u. Finally, for point 3, that $\iota\colon \mathbf{N} * \mathbf{N} \rightarrowtail \mathbf{N} \times \mathbf{N}$ is the complement of the diagonal follows by construction of pullbacks and pushouts in $\mathbf{Ass}_S(P)$. □

10.3.3 Species of Structures

We give an example of a non-trivial category with binding structure that is not a category with bindable names. This example comes from the combinatorial theory of species of structures [12]. The theory of species of structures, introduced by Joyal [58], uses functor categories to study the kinds, or species, of structure that can be put on a set of labels. Since terms with free variables are a special kind of labelled structure, it should not be too surprising that there is a connection between species of structure and functor category models of abstract syntax with binding such as those in [30] and [49]. Indeed, Tanaka [107] uses the category of species to model linear binding.

The underlying category is the presheaf topos $\mathbf{Sets}^{\mathbb{P}}$ where $\mathbb{P}$ is the category of all finite sets with bijective functions between them. The objects of $\mathbb{P}$ are thought of as sets of labels, and a presheaf A is a species of structure in the sense that it assigns to each set of labels S in $\mathbb{P}$ a set of structures $A(S)$ over these labels. The category $\mathbf{Sets}^{\mathbb{P}}$ carries a monoidal structure $\otimes$ defined as follows.

$$(A \otimes B)(S) = \{\langle S_1, a, S_2, b\rangle \mid a \in A(S_1),\ b \in B(S_2),\ S_1 \cap S_2 = \emptyset,\ S_1 \cup S_2 = S\}$$

$$(u \otimes v)_{(S_1 \cup S_2)}(\langle S_1, a, S_2, b\rangle) = \langle S_1, u_{S_1}(a), S_2, v_{S_2}(b)\rangle$$

An object X of a single abstract variable is defined by

$$X(S) = \begin{cases} S & \text{if } S \text{ is a singleton,} \\ \emptyset & \text{otherwise.} \end{cases}$$

We note that, for an element $\langle S_1, a, S_2, n\rangle \in (A \otimes X)(S)$, the sets S_1 and S_2 are uniquely determined as $S_2 = \{n\}$ and $S_1 = S \setminus \{n\}$. We will therefore omit the sets S_1 and S_2, writing just $\langle a, n\rangle$ for the elements of sets of the form $(A \otimes X)(S)$. The object $(A \otimes X)$ may be thought of as consisting of the elements of A together with a fresh variable.

The monodial structure $\otimes$ is an instance of Day's tensor and X is the representable presheaf on a singleton. We refer to [107, 108] and [12] for detailed discussions of the motivation of these definitions.

Proposition 10.3.13. *The triple* $(\mathbf{Sets}^{\mathbb{P}}, \otimes, X)$ *is a category with binding structure.*

Proof. By Lemma 2.2.9, a fibred functor is an equivalence if and only if it is full and faithful and essentially surjective on objects.

First we show that W_X is a fibred functor from the codomain fibration on $\mathbf{Sets}^{\mathbb{P}}$ to $Gl(-\otimes X)$. This means that if the square on the left is a pullback then so is the square on the right.

$$\begin{array}{ccc} A & \xrightarrow{u} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ C & \xrightarrow[v]{} & D \end{array} \qquad\qquad \begin{array}{ccc} A\otimes X & \xrightarrow{u\otimes X} & B\otimes X \\ {\scriptstyle f\otimes X}\downarrow & & \downarrow{\scriptstyle g\otimes X} \\ C\otimes X & \xrightarrow[v\otimes X]{} & D\otimes X \end{array} \tag{10.10}$$

Since pullbacks in presheaf categories are taken pointwise, it suffices to show that, at every stage S, the right square is a pullback in **Sets**. By the definitions of the tensor $\otimes$ and the object X, the elements of $(B\otimes X)(S)$ are pairs $\langle b,n\rangle$ where $n\in S$ and $b\in B(S\setminus\{n\})$. Such a pair is mapped by $(g\otimes X)_S$ to $\langle g_{S\setminus\{n\}}(b),n\rangle$. It therefore follows that any two elements $b'\in(B\otimes X)(S)$ and $c'\in(C\otimes X)(S)$ satisfying $(g\otimes X)_S(b')=(v\otimes X)_S(c')$ must be of the form $b'=\langle b,n\rangle$ and $c'=\langle c,n\rangle$ for some name $n\in S$ and some elements $b\in B(S\setminus\{n\})$ and $c\in C(S\setminus\{n\})$ satisfying $v_{S\setminus\{n\}}(c)=g_{S\setminus\{n\}}(b)$. Since the square on the left is a pullback, there exists a unique element $a\in A(S\setminus\{n\})$ satisfying $u_{S\setminus\{n\}}(a)=b$ and $f_{S\setminus\{n\}}(a)=c$. Therefore, the element $a'\stackrel{\text{def}}{=}\langle a,n\rangle\in(A\otimes X)(S)$ is the unique element of $(A\otimes X)(S)$ such that $(u\otimes X)_S(a')=b'$ and $(f\otimes X)_S(a')=c'$. This shows that the right square is a pullback.

To show that W_X is faithful, it suffices to show that $-\otimes X$ is faithful. To this end consider $u,v\colon A\to B$ and suppose $u\otimes X=v\otimes X$. By definition of the morphism action of $-\otimes X$, this implies $u_{S\setminus\{n\}}=v_{S\setminus\{n\}}$ for all stages S and all $n\in S$. Since each stage can be written as $S\setminus\{n\}$ for some n, this implies the required $u=v$.

For fullness of W_X let (u,v) be a map in $\mathbf{Sets}^{\mathbb{P}}/(-\otimes X)$ between objects that are in the image of W_X. Such a map amounts to a commuting square of the following form.

$$\begin{array}{ccc} A\otimes X & \xrightarrow{u} & B\otimes X \\ {\scriptstyle f\otimes X}\downarrow & & \downarrow{\scriptstyle g\otimes X} \\ C\otimes X & \xrightarrow[v\otimes X]{} & D\otimes X \end{array}$$

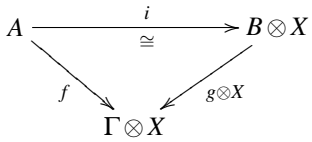
To show that W_X is full, we have to find a map $w\colon A\to B$ such that $u=w\otimes X$ and $g\circ w=v\circ f$ hold. By definition, an element of $(A\otimes X)(S)$ is a pair $\langle a,n\rangle$ where $n\in S$ and $a\in A(S\setminus\{n\})$. Because the above square commutes, this pair is mapped by u_S to an element of $(B\otimes X)(S)$ of the form $\langle b,n\rangle$ where $b\in B(S\setminus\{n\})$. Using this, we define the natural transformation w by $w_S(a)=b$ where b is the unique element of $B(S)$ such that $u_{S\cup\{n\}}$ maps $\langle a,n\rangle$ to $\langle b,n\rangle$ for some name $n\notin S$. We verify that this is well-defined, i.e. does not depend on the choice of name. Let m and n be two names not in S. We have to show $u_{S\cup\{n\}}(\langle a,n\rangle)=u_{S\cup\{m\}}(\langle a,m\rangle)$. We know that $u_{S\cup\{n\}}(\langle a,n\rangle)$ must have the form $\langle b,n\rangle$ for some b. It suffices to show $u_{S\cup\{m\}}(\langle a,m\rangle)=\langle b,m\rangle$. This equation can be shown by the following calculation, in

which we write $[m/n]\colon (S\cup\{m\}) \to (S\cup\{n\})$ for the bijection that maps m to n and all other names to themselves.

$$\begin{aligned}
u_{S\cup\{m\}}(\langle a,m\rangle) &= u_{S\cup\{m\}}([m/n]\cdot\langle a,n\rangle) && \text{by definition of the action on } \otimes\\
&= [m/n]\cdot u_{S\cup\{n\}}(\langle a,n\rangle) && \text{by naturality of } u\\
&= [m/n]\cdot\langle b,n\rangle && \text{by assumption}\\
&= \langle b,m\rangle && \text{by definition of the action on } \otimes
\end{aligned}$$

This shows that u is well-defined. Naturality of w follows from the naturality of u. That $u = w\otimes X$ holds follows directly from the definition of w. The equality $g\circ w = v\circ f$ follows because $-\otimes X$ being faithful implies that the diagram on the left in (10.10) commutes whenever the diagram on the right does.

It remains to show that W_X is essentially surjective, i.e. that each object in $\mathbf{Sets}^{\mathbb{P}}/(-\otimes X)$ is isomorphic to an object in the image of W_X. Let $f\colon A\to\Gamma\otimes X$ be an object in $\mathbf{Sets}^{\mathbb{P}}/(\Gamma\otimes X)$. We have to find an object g of $\mathbf{Sets}^{\mathbb{P}}/\Gamma$ and an isomorphism i such that the following triangle commutes.



Let δA be the presheaf defined by $\delta A = \mathbf{Sets}^{\mathbb{P}}(\mathbf{y}\{m\}, A)$, where m is an arbitrary name. By the Yoneda lemma, for any stage S and any name n not in S, we have an isomorphism $\delta A(S)\cong A(S\cup\{n\})$. For $n\notin S$ and $a\in A(S\cup\{n\})$, we write $n.a$ for the element of $\delta A(S)$ given by this isomorphism. Let B be the subobject of δA defined by $B(S) = \{n.a \mid n\notin S \wedge a\in A(S\cup\{n\}) \wedge f_{S\cup\{n\}}(a) = \langle\gamma,n\rangle\}$. Define a map $g'\colon B\otimes X\to\Gamma\otimes X$ by $g'_S(\langle n.a,m\rangle) = f_S([m/n]\cdot a)$, where $[m/n]\colon (S\setminus\{n\})\cup\{m\}\to S$ is the bijection mapping m to n and all other names to themselves. By naturality of f, this definition of g'_S is such that any pair $\langle n.a,m\rangle$ is mapped to a pair of the form $\langle\gamma,m\rangle$. As in the above argument showing that W_X is full, we can define a map $g\colon B\to\Gamma$ satisfying $g' = g\otimes X$ by letting $g_S(n.a) = \gamma$. Finally, we define $i\colon A\to B\otimes X$ by $i_S(a) = \langle n.a,n\rangle$, where n is given by $f_S(a) = \langle\gamma,n\rangle$. With these definitions, the above triangle commutes. Furthermore, i is an isomorphism with inverse $i^{-1}(\langle n.a,m\rangle) = [m/n]\cdot a$. □

That $\mathbf{Sets}^{\mathbb{P}}$ is not a category with bindable names follows immediately because $\otimes$ is not affine. A unit I for $\otimes$ is defined by $I(\emptyset) = \{\emptyset\}$ and $I(S) = \emptyset$ for all $S\neq\emptyset$, while a terminal object 1 of $\mathbf{Sets}^{\mathbb{P}}$ is defined by $1(S) = \{\emptyset\}$ for all S. Moreover, the object X could not reasonably be called an object of names, as, for example, the diagonal $X\to X\times X$ is an isomorphism.

The binding structure in the above proposition identifies the structure that Tanaka [107] uses to represent syntax with linear binders. The functor $X\multimap(-)$ that is obtained from the binding structure by means of Proposition 10.1.3 is just the shift functor δ satisfying $(\delta A)(S)\cong A(S\cup\{n\})$ for any name n not in S. We describe the binding operation $\eta'\colon (1\otimes X)\times A\to (X\multimap A)\otimes X$ that we get from Proposition 10.1.7. By unfolding definitions, it follows that there is an isomorphism $1\otimes X\cong EX$, where

EX is defined by $EX(S) = S$. The map η' thus amounts to a map of type $EX \times A \to (X \multimap A) \otimes X$. It maps a pair $\langle n,x\rangle \in (EX \times A)(S)$, where $n \in S$ and $x \in A(S)$, to a pair $\langle n.x,n\rangle \in ((X \multimap A) \otimes X)(S)$, where $n.x \in (X \multimap A)(S \setminus \{n\})$ is the element which under the isomorphism $(X \multimap A)(S \setminus \{n\}) \cong A(S)$ corresponds to x. For example, if A is the presheaf of (linear) λ-terms as in [107], then $\langle n,x\rangle$ is a pair of a term x and a variable n, and $n.x$ is the α-equivalence class of x with respect to n. The map η' is therefore just like the familiar binding operation in the Schanuel topos. In [107, 90], the emphasis is on viewing $X \multimap (-)$ as a right adjoint to $- \otimes X$, corresponding to the $\Pi^{\otimes}$-part of the binding structure. The $\Sigma^{\otimes}$-part of the binding structure, however, does not appear explicitly in [107]. Still, Tanaka uses the fact that $X \multimap (-)$ has a right adjoint to obtain that $X \multimap (-)$ preserves colimits. The existence of this right adjoint is a consequence of the above binding structure, as we observed in Proposition 10.1.3. The substitution monoidal structure studied in [107, 90, 108] and also in [30] does not, however, appear to be immediately related to our binding structure. This is most likely due to a conceptual difference, since we study the properties of binding and names, while in loc. cit. the properties of substitution for syntax with variable binders are studied.

10.4 Non-instances

We conclude this chapter with a discussion of categories that are frequently being used to model abstract syntax with binders but that are not instances of our definition of binding structure.

Fiore, Plotkin & Turi [30] and Hofmann [49] model abstract syntax with variable binders in the category $\mathbf{Sets}^{\mathbb{F}}$, where $\mathbb{F}$ is the category of finite sets and all functions between them. The monoidal structure used there is the cartesian product $\times$ and the type of variables V is the presheaf $V(S) = S$. The triple $(\mathbf{Sets}^{\mathbb{F}}, \times, V)$, however, is not a category with binding structure. This emphasises the point that a category with binding structure contains more structure than just a representation of α-equivalence classes. The category $\mathbf{Sets}^{\mathbb{F}}$ is very well-suited for representing α-equivalence classes and working with substitution. It therefore supports the convention 2.1.12 of the BVC, given at the beginning of this chapter. For many applications, supporting just this convention is sufficient, and $\mathbf{Sets}^{\mathbb{F}}$ can be used as a simple semantics for such applications. The conventions 2.1.13 and 2.1.14, however, are not satisfied by $\mathbf{Sets}^{\mathbb{F}}$. In the internal language of this category, working with α-equivalence classes is not the same as working with specific instances of these classes. The reason for this is that subobjects must be closed under arbitrary renamings, and so in particular under non-injective ones. This problem has led Hofmann [49] to give a model of the Theory of Contexts [51] in which the terms are interpreted in $\mathbf{Sets}^{\mathbb{F}}$ but the logic is interpreted in $\mathbf{Sets}^{\mathbb{I}}$, where the category $\mathbb{I}$ is the restriction of $\mathbb{F}$ to injective functions.

To see that $(\mathbf{Sets}^{\mathbb{F}}, \times, V)$ is not a category with binding structure, note first that, for the cartesian product, the simple monoidal sums and products are isomorphic to the normal sums and products, that is we have $\Sigma_V^{\times} \cong \Sigma_V$ and $\Pi_V^{\times} \cong \Pi_V$. It can then be seen that there is no isomorphism $\Sigma_V \cong \Pi_V$ because we have $(\Sigma_V V) \cong (V \times V)$ and $(\Pi_V V) \cong (V \Rightarrow V) \cong (1 + V)$, see [30], and, at stage $\{x\}$,

this gives $(V \times V)(\{x\}) = \{\langle x,x\rangle\} \ncong \{\kappa_1(\diamond), \kappa_2(x)\} \cong (1+V)(\{x\})$. Intuitively, the reason why $\mathbf{Sets}^{\mathbb{F}}$ fails to have binding structure is that is does not have a good binding operation $v.x$, that is a morphism $(-).(-)\colon V \times A \to (V \Rightarrow A)$ satisfying reasonable conditions for binding. Suppose we had such an operation in $\mathbf{Sets}^{\mathbb{F}}$. For different variables v and w we would expect $v.v$ to be different from $v.w$. The isomorphism $(V \Rightarrow V) \cong (1+V)$ is such that the term $v.v$ should correspond to the left summand in $1+V$ and the term $v.w$ should correspond to the right summand. However, in $\mathbf{Sets}^{\mathbb{F}}$ all maps must be natural with respect to all functions in $\mathbb{F}$, in particular those functions that identify v and w, and this would imply $v.v = v.w$.

Another category used frequently to model variable binders is the functor category $\mathbf{Sets}^{\mathbb{I}}$, where $\mathbb{I}$ is the category of finite sets with injections, see e.g. [49, 31]. The presheaf of variables V is defined by $V(S) = S$. The monoidal exponent $V \multimap (-)$ used to model variable binders arises by considering the monoidal structure $*$ on $\mathbf{Sets}^{\mathbb{I}}$ that is given by lifting of the disjoint sum monoidal structure on $\mathbb{I}$ by means of Day's construction [27].

$$(A * B)(S) = \int^{S_1,S_2} \mathbb{I}(S_1 + S_2, S) \times A(S_1) \times B(S_2)$$

The following two lemmas show that this monoidal structure $*$ is not well-behaved enough to make $(\mathbf{Sets}^{\mathbb{I}}, *, V)$ a category with bindable names.

Lemma 10.4.1. *There exists an object A in $\mathbf{Sets}^{\mathbb{I}}$ for which the canonical map $A * V \to A \times V$ is not monomorphic.*

Proof. Let A be the presheaf defined by

$$A(S) = \begin{cases} \{1,2\} & \text{if } S = \emptyset, \\ \{1\} & \text{otherwise,} \end{cases}$$

with morphism action defined by $A(f)(1) = 1$, $A(\emptyset \rightarrowtail \emptyset)(2) = 2$ and $A(\emptyset \rightarrowtail S)(2) = 1$, where $S \neq \emptyset$ and f is an arbitrary injection. For any variable x, the set $(A * V)(\{x\})$ defined by the above coend contains at least two different elements, namely those given by the equivalence classes $[id\colon \emptyset + \{x\} \rightarrowtail \{x\}, 1, x]$ and $[id\colon \emptyset + \{x\} \rightarrowtail \{x\}, 2, x]$. That these equivalence classes do not coincide can be seen by observing that $V(\emptyset) = \emptyset$ implies that the set $\mathbb{I}(S_1 + S_2, \{x\}) \times A(S_1) \times V(S_2)$ is non-empty if and only if S_1 is empty. The set $(A \times V)(\{x\})$, on the other hand, contains just one element, namely $\langle 1, x\rangle$. Hence, there can be no monomorphism from the set $(A * V)(\{x\})$ to $(A \times V)(\{x\})$. Since monomorphic natural transformations are pointwise monomorphisms, the map $A * V \to A \times V$ cannot be monomorphic. □

The next lemma shows that Proposition 10.2.5 does not hold in $\mathbf{Sets}^{\mathbb{I}}$.

Lemma 10.4.2. *In $\mathbf{Sets}^{\mathbb{I}}$ the quantifiers $\exists_{\dot{\pi}_1}$ and $\forall_{\dot{\pi}_1}$ along $\dot{\pi}_1\colon (-) * V \to (-)$ do not agree.*

Proof. The equality of quantifiers $\exists_{\dot{\pi}_1} = \forall_{\dot{\pi}_1}$ holds if and only if the functor $\dot{\pi}_1^*\colon Sub(A) \to Sub(A * V)$ is an isomorphism of posets for all objects A, see [72, Lemmas 2.1 and 2.2]. To see that $\dot{\pi}_1^*$ is not always

an isomorphism in $\mathbf{Sets}^{\mathbb{I}}$, consider the object A defined by

$$A(S) = \begin{cases} \emptyset & \text{if } S = \emptyset, \\ \{1\} & \text{otherwise,} \end{cases}$$

with morphism action defined by $A(f)(1) = 1$ for all injections f. We show that $\dot{\pi}_1^*$ cannot be monomorphic. Since pullbacks are taken pointwise, we have, for any subobject m of A and any stage S, the following pullback.

$$\begin{array}{ccc} \dot{\pi}_1^* B(S) & \longrightarrow & B(S) \\ {\scriptstyle \dot{\pi}_1^* m_S}\downarrow & \lrcorner & \downarrow{\scriptstyle m_S} \\ (A * V)(S) & \xrightarrow[(\pi_1)_S]{} & A(S) \end{array}$$

The set $(A * V)(\{x\})$ is empty, since, because of $A(\emptyset) = V(\emptyset) = \emptyset$, the set $\mathbb{I}(S_1 + S_2, \{x\}) \times A(S_1) \times V(S_2)$ in the coend definition of $*$ is empty. By the above pullback, this implies $\dot{\pi}_1^* B(\{x\}) = \emptyset$. Now consider the two subobjects $id \colon A \rightarrowtail A$ and $m \colon B \rightarrowtail A$, where

$$B(S) = \begin{cases} \emptyset & \text{if } |S| \leq 1, \\ \{1\} & \text{otherwise,} \end{cases}$$

and m is the evident inclusion. These two subobjects of A are mapped by $\dot{\pi}_1^*$ to the same subobject of $A * V$. This is so because the subobjects differ only in their value at stages S with $|S| = 1$, and we have seen above that, at such stages, $\dot{\pi}_1^*$ takes both subobjects to the empty set. This shows that $\dot{\pi}_1^*$ is not monomorphic and thus implies the assertion. □

The counterexamples in the above lemmas arguably show that the presheaves in $\mathbf{Sets}^{\mathbb{I}}$ do not quite model the intuition that the elements of a presheaf A at stage S consists of all the elements whose free names are contained in S. The counterexamples violate the following two reasonable requirements. First, if we take two different elements whose free names are contained in S and consider them as elements with more names $S + T$ then they should still be different. Second, suppose we have an element M whose free names are contained in $S + \{x\}$ and which has the property that, for all sets T of new names and all elements $y \in T$, we have $M = [y/x] \cdot M$ at stage $S + \{x\} + T$. Since this means that in M we can replace the name x with an arbitrary new name without changing M, it is reasonable to require that the free names of M are contained in S alone. The two reasonable requirements that we have just sketched can be enforced by restricting to the subcategory of $\mathbf{Sets}^{\mathbb{I}}$ consisting of sheaves with respect to the atomic topology on $\mathbb{I}^{\mathrm{op}}$. The resulting category is equivalent to the Schanuel topos.

Although the above two lemmas show that $(\mathbf{Sets}^{\mathbb{I}}, *, V)$ is not a category with bindable names, it could still be a category with binding structure. However, the only description of the monoidal structure $*$ we have is the above coend. This description is hard to work with, which makes proving or disproving that $(\mathbf{Sets}^{\mathbb{I}}, *, V)$ is a category with binding structure quite complicated. This is another reason for preferring the Schanuel topos to $\mathbf{Sets}^{\mathbb{I}}$: the monoidal structure $*$ is much easier to work with.

As in the case for $\mathbf{Sets}^{\mathbb{F}}$, that $\mathbf{Sets}^{\mathbb{I}}$ is not a category with bindable names is not a problem for applications that need only part of structure of a category with bindable names. In $\mathbf{Sets}^{\mathbb{I}}$, α-equivalence classes can be represented using Π^* and subobjects need only be closed under injective renaming. This is sufficient for many applications, e.g. as in [49].

10.5 Further Work

One obvious direction for further work is to look for more instances of categories with binding structure and with bindable names. One likely candidate is a version of the Schanuel topos in which the countably infinite object of names is replaced by an uncountable one and in which elements are allowed to have countable support. This category has an object of infinite streams of pairwise distinct names, as studied in [36]. Since this object has essentially simple transitive action, it is reasonable to expect it to form part of a binding structure. There are many other categories that may also be categories with binding structure or with bindable names, such as the categories of containers of Abbott et al. [1] or the domain theoretic categories of Shinwell & Pitts [101].

Beyond looking for more instances, we think that the general theory of categories with binding structure deserves further study. An interesting question is whether it is possible to freely add binding structure or bindable names to a given category. For example, it may be the case that the Schanuel topos can be constructed by freely adding bindable names to the topos of sets. Perhaps the construction of the models in Chapter 3 can serve as a starting point for such a free construction. Apart from theoretical interest, such a construction would have interesting applications. For instance, if we could add bindable names to the term model of a dependent type theory, then the interpretation of our type theory in this model would amount to a translation of the bunched type theory with names into normal type theory.

Another interesting direction for further work is to study how categories with bindable names relate to FM-style approaches that are based on swapping. In the next chapter we give some evidence that categories with bindable names contain a swapping operation. However, we are not yet able to prove all the equations such a swapping operation should satisfy. For further work, we ask the question whether or not any category with bindable names contains a natural transformation $\sigma : (\mathbf{N} \times \mathbf{N}) \times (-) \to (-)$ that satisfies the following equations for all generalised elements $m, n \colon \mathbf{N}$, $x \colon A$ and $y \colon B$.

$$\begin{aligned}
\sigma_A\langle\langle m,m\rangle,x\rangle &= x \\
\sigma_A\langle\langle m,n\rangle,x\rangle &= \sigma_A\langle\langle n,m\rangle,x\rangle \\
\sigma_A\langle\langle m,n\rangle,\sigma_A\langle\langle m,n\rangle,x\rangle\rangle &= x \\
\sigma_{\mathbf{N}}\langle\langle m,n\rangle,m\rangle &= n \\
\sigma_{A\times B}\langle\langle m,n\rangle,\langle x,y\rangle\rangle &= \langle\sigma_A\langle\langle m,n\rangle,x\rangle,\sigma_B\langle\langle m,n\rangle,y\rangle\rangle \\
\sigma_A \circ \iota_{(\mathbf{N}\times\mathbf{N}),A} &= \hat{\pi}_2 : (\mathbf{N}\times\mathbf{N}) * A \to A
\end{aligned}$$

In the Schanuel topos and in $\mathbf{Ass}_S(P)$, σ can be defined by the swapping action $\sigma_A\langle\langle m,n\rangle,x\rangle = (m\ n)\cdot x$.

Chapter 11

Type Theory with Names and Binding

In the chapter on categories with bindable names we have shown how name binding can be expressed in terms of simple monoidal sums Σ^* and products Π^*. On the basis of this definition, we now extend the syntax of the type theory with constructs for names and name-binding. We add a type of names and also hidden-name types, which make available the structure of categories with bindable names.

11.1 Names

First, we extend the type theory with a type of names $\mathbf{N}$ having decidable equality. We write short **BTN** for the resulting type theory $\mathbf{BT}(*, 1, \Sigma, \Pi, \Pi^*, B^{*(M:A)}, \Sigma^*, \mathbf{N})$.

$$(\mathbf{N}\text{-TY})\ \frac{}{\vdash \mathbf{N}\ \mathrm{Type}}$$

$$(\text{ifeq-I})\ \frac{\begin{array}{ll}\Gamma \vdash P : \mathbf{N}\times\mathbf{N} & \Gamma, n\colon \mathbf{N} \vdash M : A[\langle n,n\rangle/p] \\ \Gamma, p\colon \mathbf{N}\times\mathbf{N} \vdash A\ \mathrm{Type} & \Gamma, q\colon \mathbf{N}*\mathbf{N} \vdash N : A[\iota(q)/p]\end{array}}{\Gamma \vdash \text{ifeq } P \text{ then } n.M \text{ else } q.N : A[P/p]}$$

$$(\text{ifeq-THEN})\ \frac{\begin{array}{ll}\Gamma \vdash R : \mathbf{N} & \Gamma, n\colon \mathbf{N} \vdash M : A[\langle n,n\rangle/p] \\ \Gamma, p\colon \mathbf{N}\times\mathbf{N} \vdash A\ \mathrm{Type} & \Gamma, q\colon \mathbf{N}*\mathbf{N} \vdash N : A[\iota(q)/p]\end{array}}{\Gamma \vdash \text{ifeq } \langle R,R\rangle \text{ then } n.M \text{ else } q.N = (M[R/n]) : A[\langle R,R\rangle/p]}$$

$$(\text{ifeq-ELSE})\ \frac{\begin{array}{ll}\Gamma \vdash Q : \mathbf{N}*\mathbf{N} & \Gamma, n\colon \mathbf{N} \vdash M : A[\langle n,n\rangle/p] \\ \Gamma, p\colon \mathbf{N}\times\mathbf{N} \vdash A\ \mathrm{Type} & \Gamma, q\colon \mathbf{N}*\mathbf{N} \vdash N : A[\iota(q)/p]\end{array}}{\Gamma \vdash \text{ifeq } \iota(Q) \text{ then } n.M \text{ else } q.N = (N[Q/q]) : A[\iota(Q)/p]}$$

We write short $(\text{ifeq } P \text{ then } M \text{ else } N)$ for $(\text{ifeq } P \text{ then } n.M \text{ else } q.N)$ if $n \notin FV(M)$ and $q \notin FV(N)$ hold. We omit the evident congruence rules.

11.1.1 Example

To give an example for the use of comparable names, we define a function that removes a name from a list of names. It type is

$$\vdash \mathsf{remove} : \Pi n\colon \mathbf{N}.\, \mathsf{LN} \to \left(\mathsf{LN}^{*(n:\mathbf{N})}\right),$$

where ($\vdash \mathsf{LN}$ Type) is an inductively defined type of lists of names. The freshness information in the type of remove already contains the information that the name n does not appear in the result list. In Section 12.1 we make essential use of the freshness information in the type of remove when we define a function that computes the free variables of a term. There the freshness information will be needed in order to apply rule (Σ^*-E).

The type LN is defined inductively with two constructors

$$\vdash \mathsf{nil} : \mathsf{LN}, \qquad \vdash \mathsf{cons} : \mathbf{N} \to \mathsf{LN} \to \mathsf{LN},$$

and the standard recursion principle

$$(\mathsf{LN\text{-}REC})\ \frac{\begin{array}{l}\Gamma, x\colon \mathsf{LN} \vdash A\ \mathrm{Type}\\ \Gamma \vdash M : A[\mathsf{nil}/x]\\ \Gamma \vdash N : \Pi n\colon \mathbf{N}.\, \Pi y\colon \mathsf{LN}.\, A[y/x] \to A[\mathsf{cons}\ n\ y/x]\end{array}}{\Gamma \vdash \mathsf{rec}_A(M,N) : \Pi x\colon \mathsf{LN}.\, A}$$

with the standard equations.

The term remove is defined by induction on its second argument. In the recursion principle for LN we take the type A to be ($n\colon \mathbf{N} \vdash \mathsf{LN}^{*(n:\mathbf{N})}$ Type), omitting the unused variable x. Given the empty list as argument, (remove n) should return the empty list. Therefore, we define the base case by the term in the conclusion of the following derivation.

$$\dfrac{\dfrac{\vdash \mathsf{nil} : \mathsf{LN} \qquad \dfrac{\vdash \mathbf{N}\ \mathrm{Type}}{n\colon \mathbf{N} \vdash n : \mathbf{N}}}{\diamond * n\colon \mathbf{N} \vdash \mathsf{nil}^{*n} : \mathsf{LN}^{*(n:\mathbf{N})}}}{n\colon \mathbf{N} \vdash \mathsf{nil}^{*n} : \mathsf{LN}^{*(n:\mathbf{N})}}$$

Given the argument (cons m y), the function (remove n) should return (cons m (remove n y)) if $m \neq n$ holds and (remove n y) if $m = n$ holds. Of course, the terms in this informal explanation are not typeable, because the type of remove contains a freefrom type. We start the precise derivation of the recursion step from the root as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}n\colon \mathbf{N}, m\colon \mathbf{N}, y\colon \mathsf{LN}, r\colon \mathbf{N} \vdash M : \mathsf{LN}^{*(\mathrm{fst}(\langle r,r\rangle):\mathbf{N})} \to \mathsf{LN}^{*(\mathrm{snd}(\langle r,r\rangle):\mathbf{N})}\\ n\colon \mathbf{N}, m\colon \mathbf{N}, y\colon \mathsf{LN}, q\colon \mathbf{N}*\mathbf{N} \vdash N : \mathsf{LN}^{*(\pi_1(q):\mathbf{N})} \to \mathsf{LN}^{*(\pi_1(q):\mathbf{N})}\end{array}}{n\colon \mathbf{N}, m\colon \mathbf{N}, y\colon \mathsf{LN} \vdash \mathsf{ifeq}\ \langle n,m\rangle\ \mathsf{then}\ r.M\ \mathsf{else}\ q.N : \mathsf{LN}^{*(\mathrm{fst}(\langle n,m\rangle):\mathbf{N})} \to \mathsf{LN}^{*(\mathrm{snd}(\langle n,m\rangle):\mathbf{N})}}}{n\colon \mathbf{N}, m\colon \mathbf{N}, y\colon \mathsf{LN} \vdash \mathsf{ifeq}\ \langle n,m\rangle\ \mathsf{then}\ r.M\ \mathsf{else}\ q.N : \mathsf{LN}^{*(n:\mathbf{N})} \to \mathsf{LN}^{*(n:\mathbf{N})}}}{n\colon \mathbf{N}, m\colon \mathbf{N} \vdash \lambda y\colon \mathsf{LN}.\, \mathsf{ifeq}\ \langle n,m\rangle\ \mathsf{then}\ r.M\ \mathsf{else}\ q.N : \mathsf{LN} \to \mathsf{LN}^{*(n:\mathbf{N})} \to \mathsf{LN}^{*(n:\mathbf{N})}}}{n\colon \mathbf{N} \vdash \lambda m\colon \mathbf{N}.\, \lambda y\colon \mathsf{LN}.\, \mathsf{ifeq}\ \langle n,m\rangle\ \mathsf{then}\ r.M\ \mathsf{else}\ q.N : \mathbf{N} \to \mathsf{LN} \to \mathsf{LN}^{*(n:\mathbf{N})} \to \mathsf{LN}^{*(n:\mathbf{N})}}$$

The term M corresponds to the case where $n = m$ and N corresponds to the case where $n \neq m$. As described informally above, M should be the identity.

$$M \stackrel{\text{def}}{=} \lambda h\colon \mathsf{LN}^{*(\mathrm{fst}(\langle r,r\rangle):\mathbf{N})}.\, h.$$

We come to the definition of N. Because there are derivations

$$\dfrac{\dfrac{\dfrac{m\colon \mathbf{N} \vdash m : \mathbf{N} \qquad n\colon \mathbf{N} \vdash n : \mathbf{N}}{m\colon \mathbf{N} * n\colon \mathbf{N} \vdash m^{*n} : \mathbf{N}^{*(n:\mathbf{N})}}}{(m\colon \mathbf{N} * n\colon \mathbf{N}), h\colon \mathsf{LN}^{*(n:\mathbf{N})} \vdash m^{*n} : \mathbf{N}^{*(n:\mathbf{N})}} \qquad (m\colon \mathbf{N} * n\colon \mathbf{N}), h\colon \mathsf{LN}^{*(n:\mathbf{N})} \vdash h : \mathsf{LN}^{*(n:\mathbf{N})}}{(m\colon \mathbf{N} * n\colon \mathbf{N}), h\colon \mathsf{LN}^{*(n:\mathbf{N})} \vdash \mathrm{join}(m^{*n}, h) : (\mathbf{N} \times \mathsf{LN})^{*(n:\mathbf{N})}}$$

and

$$\dfrac{m\colon \mathbf{N}, x\colon \mathsf{LN} \vdash \mathsf{cons}\ m\ x : \mathsf{LN} \qquad n\colon \mathbf{N} \vdash n : \mathbf{N}}{(m\colon \mathbf{N}, x\colon \mathsf{LN}) * n\colon \mathbf{N} \vdash (\mathsf{cons}\ m\ x)^{*n} : \mathsf{LN}^{*(n:\mathbf{N})}}$$

we can use (FF-E) to derive

$$\dfrac{(n\colon \mathbf{N} * m\colon \mathbf{N}), h\colon \mathsf{LN}^{*(n:\mathbf{N})} \vdash \mathrm{let}\ \mathrm{join}(m^{*n}, h)\ \mathrm{be}\ \langle m, x\rangle^{*n}\ \mathrm{in}\ (\mathsf{cons}\ m\ x)^{*n} : \mathsf{LN}^{*(n:\mathbf{N})}}{n\colon \mathbf{N} * m\colon \mathbf{N} \vdash \lambda h\colon \mathsf{LN}^{*(n:\mathbf{N})}.\, \mathrm{let}\ \mathrm{join}(m^{*n}, h)\ \mathrm{be}\ \langle m, x\rangle^{*n}\ \mathrm{in}\ (\mathsf{cons}\ m\ x)^{*n} : \mathsf{LN}^{*(n:\mathbf{N})} \to \mathsf{LN}^{*(n:\mathbf{N})}}$$

(for convenience we use the pattern $\langle m, x\rangle$ rather than projections in this term). We write N' as an abbreviation for the term in the conclusion of this derivation. The term N' is essentially the definition of N, we just have to put it in the right context. This can be done as follows.

$$\dfrac{\dfrac{\dfrac{n\colon \mathbf{N} * m\colon \mathbf{N} \vdash N' : \mathsf{LN}^{*(n:\mathbf{N})} \to \mathsf{LN}^{*(n:\mathbf{N})}}{n\colon \mathbf{N} * m\colon \mathbf{N} \vdash N' : \mathsf{LN}^{*(\dot\pi_1(n*m):\mathbf{N})} \to \mathsf{LN}^{*(\dot\pi_1(n*m):\mathbf{N})}}}{q\colon \mathbf{N} * \mathbf{N} \vdash \mathrm{let}\ q\ \mathrm{be}\ n*m\ \mathrm{in}\ N' : \mathsf{LN}^{*(\dot\pi_1(q):\mathbf{N})} \to \mathsf{LN}^{*(\dot\pi_1(q):\mathbf{N})}}}{n\colon \mathbf{N}, m\colon \mathbf{N}, y\colon \mathsf{LN}, q\colon \mathbf{N} * \mathbf{N} \vdash \mathrm{let}\ q\ \mathrm{be}\ n*m\ \mathrm{in}\ N' : \mathsf{LN}^{*(\dot\pi_1(q):\mathbf{N})} \to \mathsf{LN}^{*(\dot\pi_1(q):\mathbf{N})}}$$

We define N to be the term in the conclusion of this derivation.

With these definitions of the base and recursion cases for the definition of remove, we have

$$\dfrac{n\colon \mathbf{N} \vdash \mathsf{rec}_{(\mathsf{LN}^{*(n:\mathbf{N})})}(\mathsf{nil}^{*n}, \lambda m\colon \mathbf{N}.\, \lambda y\colon \mathsf{LN}.\, \mathrm{ifeq}\ \langle n, m\rangle\ \mathrm{then}\ r.M\ \mathrm{else}\ q.N) : \mathsf{LN} \to \mathsf{LN}^{*(n:\mathbf{N})}}{\vdash \lambda n\colon \mathbf{N}.\, \mathsf{rec}_{(\mathsf{LN}^{*(n:\mathbf{N})})}(\mathsf{nil}^{*n}, \lambda m\colon \mathbf{N}.\, \lambda y\colon \mathsf{LN}.\, \mathrm{ifeq}\ \langle n, m\rangle\ \mathrm{then}\ r.M\ \mathrm{else}\ q.N) : \Pi n\colon \mathbf{N}.\, \mathsf{LN} \to \mathsf{LN}^{*(n:\mathbf{N})}}$$

and we define remove to be the term in the conclusion.

To give an example of an evaluation of remove, we show the simple equation

$$m\colon \mathbf{N} * n\colon \mathbf{N} \vdash (\mathsf{remove}\ n\ (\mathsf{cons}\ m\ (\mathsf{cons}\ n\ \mathsf{nil})) = (\mathsf{cons}\ m\ \mathsf{nil})^{*n} : \mathsf{LN}^{*(n:\mathbf{N})}.$$

In context $m\colon \mathbf{N} * n\colon \mathbf{N}$, we have the following chain of equations.

$$
\begin{aligned}
&\mathsf{remove}\ n\ (\mathsf{cons}\ m\ (\mathsf{cons}\ n\ \mathsf{nil})) \\
&= (\text{ifeq } \langle n,m\rangle \text{ then } r.M \text{ else } q.N)\ (\mathsf{remove}\ n\ (\mathsf{cons}\ n\ \mathsf{nil})) && \text{by defn. and rec-eqn.} \\
&= (\text{ifeq } \langle n,m\rangle \text{ then } r.M \text{ else } q.N)\ (\mathsf{nil}^{*n}) && \text{see below} \\
&= (\text{ifeq } \iota(n{*}m) \text{ then } r.M \text{ else } q.N)\ (\mathsf{nil}^{*n}) && \text{by } (*\text{-}\beta) \\
&= (N[n{*}m/q])\ (\mathsf{nil}^{*n}) && \text{by (ifeq-ELSE)} \\
&= (\text{let } n{*}m \text{ be } n{*}m \text{ in } N')\ (\mathsf{nil}^{*n}) && \text{by defn.} \\
&= N'\ (\mathsf{nil}^{*n}) && \text{by } (*\text{-}\beta) \\
&= (\lambda h\colon \mathsf{LN}^{*(n:\mathbf{N})}.\,\text{let join}(m^{*n},h) \text{ be } \langle m,x\rangle^{*n} \text{ in } (\mathsf{cons}\ m\ x)^{*n})\ (\mathsf{nil}^{*n}) && \text{by defn.} \\
&= \text{let join}(m^{*n},\mathsf{nil}^{*n}) \text{ be } \langle m,x\rangle^{*n} \text{ in } (\mathsf{cons}\ m\ x)^{*n} && \text{by } (\Pi\text{-}\beta)
\end{aligned}
$$

Now we show that, also in context $m\colon \mathbf{N} * n\colon \mathbf{N}$, the equality $\text{join}(m^{*n},\mathsf{nil}^{*n}) = \langle m,\mathsf{nil}\rangle^{*n}$ holds. Note that both sides are typeable. We show this equality using the rule (FF-INJECT), whose applicability can be seen as follows.

$$
\begin{aligned}
&\text{let join}(m^{*n},\mathsf{nil}^{*n}) \text{ be } z^{*n}\colon (\mathbf{N}\times\mathsf{LN})^{*(n:\mathbf{N})} \text{ in } z \\
&= \text{let join}(m^{*n},\mathsf{nil}^{*n}) \text{ be } z^{*n}\colon (\mathbf{N}\times\mathsf{LN})^{*(n:\mathbf{N})} \text{ in } \langle \text{fst}(z),\text{snd}(z)\rangle && \text{by } (\Sigma\text{-}\eta) \\
&= \langle \text{let join}(m^{*n},\mathsf{nil}^{*n}) \text{ be } z^{*n} \text{ in fst}(z), \text{let join}(m^{*n},\mathsf{nil}^{*n}) \text{ be } z^{*n} \text{ in snd}(z)\rangle && \text{by CC} \\
&= \langle m,\mathsf{nil}\rangle && \text{by (FF-JOIN-EQ1/2)} \\
&= \text{let } \langle m,\mathsf{nil}\rangle^{*n} \text{ be } z^{*n}\colon (\mathbf{N}\times\mathsf{LN})^{*(n:\mathbf{N})} \text{ in } z && \text{by (FF-}\beta\text{)}
\end{aligned}
$$

Since using (FF-INJECT) we obtain $\text{join}(m^{*n},\mathsf{nil}^{*n}) = \langle m,\mathsf{nil}\rangle^{*n}$, we can continue the calculation of $(\mathsf{remove}\ n\ (\mathsf{cons}\ m\ (\mathsf{cons}\ n\ \mathsf{nil}))) = (\mathsf{cons}\ m\ \mathsf{nil})^{*n}$ as follows.

$$
\begin{aligned}
&\text{let join}(m^{*n},\mathsf{nil}^{*n}) \text{ be } \langle m,x\rangle^{*n} \text{ in } (\mathsf{cons}\ m\ x)^{*n} \\
&= \text{let } \langle m,\mathsf{nil}\rangle^{*n} \text{ be } \langle m,x\rangle^{*n} \text{ in } (\mathsf{cons}\ m\ x)^{*n} && \text{by congruence} \\
&= (\mathsf{cons}\ m\ \mathsf{nil})^{*n} && \text{(FF-}\beta\text{)}
\end{aligned}
$$

As this is the required value, it just remains to show the equation $(\mathsf{remove}\ n\ (\mathsf{cons}\ m\ \mathsf{nil})) = \mathsf{nil}^{*n}$ that we have used above. We calculate as follows.

$$
\begin{aligned}
&\mathsf{remove}\ n\ (\mathsf{cons}\ n\ \mathsf{nil}) \\
&= (\text{ifeq } \langle n,n\rangle \text{ then } r.M \text{ else } q.N)\ (\mathsf{remove}\ n\ \mathsf{nil}) && \text{by defn.}
\end{aligned}
$$

We use the equation $(\mathsf{remove}\ n\ \mathsf{nil}) = \mathsf{nil}^{*n}$, which is immediate by definition.

$$
\begin{aligned}
&= (\text{ifeq } \langle n,n\rangle \text{ then } r.M \text{ else } q.N)\ (\mathsf{nil}^{*n}) \\
&= (M[n/r])\ (\mathsf{nil}^{*n}) && \text{by (ifeq-THEN)} \\
&= \mathsf{nil}^{*n} && \text{by defn. and } (\Pi\text{-}\beta)
\end{aligned}
$$

We have thus shown the asserted equation.

The definition of remove demonstrates a number of features of the type theory. First, the derivation of $n\colon \mathbf{N} \vdash \mathsf{nil}^{*n} : \mathsf{LN}^{*(n:\mathbf{N})}$ shows that a closed constant is provably fresh for any name. Second, the constant join is used essentially in the derivation. As a consequence, we have to deal with the equations for join in order to establish equations of remove. In the above example, we have made essential use of (FF-INJECT) in order to obtain $\mathsf{join}(m^{*n}, \mathsf{nil}^{*n}) = \langle m, \mathsf{nil}\rangle^{*n}$. Since semantically the rule (FF-INJECT) amounts to the monoidal structure $*$ being strict affine, this further motivates our focus on strict affine monoidal structures in the definition of the type theory. Lastly, the definition of remove shows how freefrom types can be used to establish a freshness property inductively. We could not have defined the term remove of that type without using freefrom types to establish a freshness assertion inductively.

11.2 Interpretation

For the interpretation of names we assume the structure of a category with bindable names. Furthermore, in order to be able to work with the isomorphism $[\delta, \iota]\colon \mathbf{N} + (\mathbf{N} * \mathbf{N}) \to \mathbf{N} \times \mathbf{N}$ from Proposition 10.2.2, we assume that the semantics models strong binary coproducts [56, §10.5]. Strong binary coproducts can, in a fibration equivalent to a codomain fibration, be constructed from universal coproducts $+$ on the base category [56, §10.5]. We have shown in Chapter 3 that both the Schanuel topos and $\mathbf{Ass}_S(P)$ have this structure.

The type $\mathbf{N}$ is interpreted by the object of names in the category with bindable names. For the interpretation of the term ifeq, we make use of the isomorphism $[\delta, \iota]\colon \mathbf{N} + (\mathbf{N} * \mathbf{N}) \to \mathbf{N} \times \mathbf{N}$. This map corresponds to the substitution

$$\langle \text{unpack } z \text{ as } (\kappa_1(n) \text{ in } \langle n, n\rangle, \kappa_2(q) \text{ in } \iota(q))/p\rangle\colon (z\colon \mathbf{N} + (\mathbf{N} * \mathbf{N})) \to (p\colon \mathbf{N} \times \mathbf{N}),$$

in which we use the notation of [56, §10.5] for binary coproducts. Write now $R(z)$ for the term $(\text{unpack } z \text{ as } (\kappa_1(n) \text{ in } \langle n, n\rangle, \kappa_2(q) \text{ in } \iota(q)))$ in this substitution. The equations for binary coproducts give us $R(\kappa_1(n)) = \langle n, n\rangle$ and $R(\kappa_2(q)) = \iota(q)$. The interpretation of the rule (ifeq-I) is then given by the following derivation.

$$\dfrac{\dfrac{\Gamma, p\colon \mathbf{N} \times \mathbf{N} \vdash A \text{ Type} \qquad \dfrac{\Gamma, n\colon \mathbf{N} \vdash M : A[\langle n, n\rangle/p]}{\Gamma, n\colon \mathbf{N} \vdash M : A[R(\kappa_1(n))/p]} \qquad \dfrac{\Gamma, q\colon \mathbf{N} * \mathbf{N} \vdash N : A[\iota(q)/p]}{\Gamma, q\colon \mathbf{N} * \mathbf{N} \vdash N : A[R(\kappa_2(q))/p]}}{\Gamma, z\colon \mathbf{N} + (\mathbf{N} * \mathbf{N}) \vdash \text{unpack } z \text{ as } (\kappa_1(n) \text{ in } M, \kappa_2(q) \text{ in } N) : A[R(z)/p]}}{\Gamma, p\colon \mathbf{N} \times \mathbf{N} \vdash \text{ifeq } P \text{ then } n.M \text{ else } q.N : A}$$

The last step in this derivation is given by reindexing along the inverse of $[\delta, \iota]$. This means, that the term $(\text{ifeq } P \text{ then } n.M \text{ else } q.N)$ is defined as the term $(\text{unpack } z \text{ as } (\kappa_1(n) \text{ in } M, \kappa_2(q) \text{ in } N))$ reindexed along $[\delta, \iota]^{-1}$. With this definition it is straightforward to establish the equations for ifeq.

11.3 Hidden-name Types

The main feature of a category with bindable names is the definition of name-binding in terms of an equivalence of fibrations. In Chapter 10, we have seen that this equivalence amounts to a certain isomorphism $i\colon \Sigma^*_{\mathbf{N}} \to \Pi^*_{\mathbf{N}}$. In this section, we present *hidden-name types* as a syntax for this isomorphism and thus for name-binding.

Because of the isomorphism, the elements of $\Sigma^* n\colon \mathbf{N}.A$ (respectively $\Pi^* n\colon \mathbf{N}.A$) can be viewed both as pairs with hidden names, i.e. elements of $\Sigma^* n\colon \mathbf{N}.A$, and as partial functions defined just on fresh names, i.e. elements of $\Pi^* n\colon \mathbf{N}.A$. Hidden-name types make available this dual view. The elements of the hidden-name type $\mathrm{H}n.A$ may be viewed both as elements of $\Sigma^* n\colon \mathbf{N}.A$ and $\Pi^* n\colon \mathbf{N}.A$. This duality of H-types is implemented by giving them the rules from both $\Sigma^*_{\mathbf{N}}$ and $\Pi^*_{\mathbf{N}}$.

We write $\mathbf{BTN}(\mathrm{H})$ for the type theory **BTN** with hidden-name types, as defined by the following rules and equations.

11.3.1 Rules for Hidden-name Types

Formation

$$(\text{H-T}_\text{Y})\ \frac{\Gamma * n\colon \mathbf{N} \vdash A\ \mathrm{Type}}{\Gamma \vdash \mathrm{H}n.A\ \mathrm{Type}}$$

Introduction

$$(\text{H-I-}\Pi^*)\ \frac{\Gamma * n\colon \mathbf{N} \vdash M : A}{\Gamma \vdash \lambda^{\mathrm{H}} n.M : \mathrm{H}n.A}$$

$$(\text{H-I-}\Sigma^*)\ \frac{n\colon \mathbf{N} \vdash A\ \mathrm{Type} \qquad \Gamma \vdash M : \mathbf{N} \qquad \Gamma \vdash N : A[M/n]}{\Gamma \vdash \mathrm{bind}^{\mathrm{H}}_{(n)A}(M,N) : (\mathrm{H}n.A)^{*(M:\mathbf{N})}}$$

Elimination

$$(\text{H-E-}\Pi^*)\ \frac{\Gamma \vdash M : \mathrm{H}n.A \qquad \Delta \vdash N : \mathbf{N}}{\Gamma * \Delta \vdash \mathrm{app}^{\mathrm{H}}_{(n)A}(M,N) : A[N/n]}$$

$$(\text{H-E-}\Sigma^*)\ \frac{\Gamma \vdash M : \mathrm{H}n.A \qquad (\Gamma * n\colon \mathbf{N}), x\colon A \vdash N : B^{*(n:\mathbf{N})}}{\Gamma \vdash \mathrm{let}\ M\ \mathrm{be}\ n.x\colon \mathrm{H}n.A\ \mathrm{in}\ N : B}$$

The isomorphism $i\colon \Sigma^*_{\mathbf{N}} \to \Pi^*_{\mathbf{N}}$ is built into the terms of these rules. For example, we can interpret $\mathrm{H}n.A$ as $\Sigma^* n\colon \mathbf{N}.A$. Then, the terms $\mathrm{bind}^{\mathrm{H}}_{(n)A}(M,N)$ and $(\mathrm{let}\ M\ \mathrm{be}\ n.x\colon \mathrm{H}n.A\ \mathrm{in}\ N)$ are interpreted as the corresponding terms for Σ^*-types, while the terms $\lambda^{\mathrm{H}} n.M$ and $\mathrm{app}^{\mathrm{H}}_{(n)A}(M,N)$ are interpreted as $i^{-1}(\lambda^* n\colon \mathbf{N}.M)$ and $\mathrm{app}^*_{(n:\mathbf{N})A}(i(M),N)$ respectively. Just as well, we can interpret $\mathrm{H}n.A$ as $\Pi^* n\colon \mathbf{N}.A$, but then the isomorphism is built into the terms for Σ^*.

In categorical terms, this definition of hidden-name types corresponds to the definition of a functor H with fibred adjunctions $\mathrm{H} \dashv W_{\mathbf{N}} \dashv \mathrm{H}$ in Proposition 10.1.2.

11.3.1.1 Equations for Hidden-name Types

The equations for hidden-name types must explain the behaviour of the introduction and elimination terms. When looked at in isolation, the terms coming from Π^* (respectively Σ^*) should behave as they do for Π^*-types (respectively Σ^*-types). Therefore, we add the equations from both Π^* and Σ^* to the equations for H-types. We refer to them as (H-β-Π^*), (H-η-Π^*), (H-β-Σ^*) and (H-η-Σ^*) but do not repeat them here. Furthermore, hidden-name types allow the introduction and elimination terms from Π^* and Σ^* to be mixed. Therefore, we have to add equations explaining such mixed use of Π^* and Σ^*. These interaction equations are precisely what the commuting triangles in Proposition 10.1.2 amount to.

Interaction Equations

Equations from the left triangle in 10.1.2

$$(\text{H-}\beta\text{-}\Sigma^*\Pi^*)\ \frac{n\colon \mathbf{N} \vdash A\ \text{Type} \qquad \Gamma \vdash N : \mathbf{N} \qquad \Gamma \vdash M : A[N/n]}{\Gamma \vdash \text{let bind}^{\mathrm{H}}(N,M)\ \text{be}\ x^{*n}\ \text{in}\ x@_{\mathrm{H}}n = M : A[N/n]}$$

$$(\text{H-}\eta\text{-}\Sigma^*\Pi^*)\ \frac{n\colon \mathbf{N} \vdash A\ \text{Type} \qquad \Gamma \vdash N : \mathbf{N} \qquad \Gamma \vdash M : (\mathrm{H}n.A)^{*(N:\mathbf{N})}}{\Gamma \vdash M = \text{bind}^{\mathrm{H}}(N, \text{let}\ M\ \text{be}\ x^{*n}\ \text{in}\ x@_{\mathrm{H}}n) : (\mathrm{H}n.A)^{*(N:\mathbf{N})}}$$

Equations from the right triangle in 10.1.2

$$(\text{H-}\beta\text{-}\Pi^*\Sigma^*)\ \frac{\vdash A\ \text{Type} \qquad \Gamma \vdash M : A}{\Gamma \vdash \text{let}\ \lambda^{\mathrm{H}}n.M^{*n}\ \text{be}\ n.x\ \text{in}\ x = M : A}$$

$$(\text{H-}\eta\text{-}\Pi^*\Sigma^*)\ \frac{\vdash A\ \text{Type} \qquad \Gamma \vdash M : \mathrm{H}n.A^{*(n:\mathbf{N})}}{\Gamma \vdash M = \lambda^{\mathrm{H}}n.(\text{let}\ M\ \text{be}\ n.x\ \text{in}\ x)^{*n} : \mathrm{H}n.A^{*(n:\mathbf{N})}}$$

We remark that if we forget about freefrom types, (H-β-$\Sigma^*\Pi^*$) and (H-η-$\Sigma^*\Pi^*$) amount to the equations $(n.x)@n = x$ and $n.(x@n) = x$, which appear as (43) and (47) in [38].

Equations from the naturality of $i\colon \Sigma^*_{\mathbf{N}} \to \Pi^*_{\mathbf{N}}$

Finally, we add the following equation deriving from the naturality of the isomorphism i.

$$(\text{H-NAT})\ \frac{\Gamma \vdash M : \mathrm{H}n.A \qquad (\Gamma * n\colon \mathbf{N}), x\colon A \vdash N : B \qquad (\Gamma * n\colon \mathbf{N}), y\colon B \vdash R : C^{*(n:\mathbf{N})}}{\Gamma \vdash \text{let}\ (\lambda^{\mathrm{H}}n.N[M@_{\mathrm{H}}n/x])\ \text{be}\ n.y\ \text{in}\ R = \text{let}\ M\ \text{be}\ n.x\ \text{in}\ R[N/y] : C}$$

Adding an equation for the naturality of i appears to be necessary, as functoriality of H is available in the syntax in two seemingly unrelated ways, namely both with the Π^*-view and the Σ^*-view of H-types.

We remark that (H-NAT) is not the most natural way of stating the naturality of the isomorphism. A more natural choice would be the following equation.

$$(\text{H-NAT}')\ \frac{\Gamma \vdash M : \mathrm{H}n.A \qquad n\colon \mathbf{N} \vdash B\ \text{Type} \qquad (\Gamma * n\colon \mathbf{N}), x\colon A \vdash N : B}{\Gamma \vdash \lambda^{\mathrm{H}}n.N[M@_{\mathrm{H}}n/x] = (\text{let}\ M\ \text{be}\ n.x\ \text{in bind}^{\mathrm{H}}(n,N)) : \mathrm{H}n.B}$$

Even though this rule is more natural, the rule (H-NAT) is more useful in applications. It is likely that, due to the restriction to closed freefrom types, the rule (H-NAT) is not derivable from (H-NAT′). Note, for example, the assumption $n\colon \mathbf{N} \vdash B$ Type in (H-NAT′). Ideally, we would like to replace it with the more general $\Gamma * n\colon \mathbf{N} \vdash B$ Type, but then the right-hand side of the equation would not be typeable. With open freefrom types, however, rule (H-NAT) would be derivable as follows.

$$\begin{aligned}
&\text{let } (\lambda^{\mathrm{H}} n.N[M@_{\mathrm{H}} n/x]) \text{ be } n.y \text{ in } R \\
&= \text{let } (\text{let } M \text{ be } n.x \text{ in } \mathrm{bind}^{\mathrm{H}}(n,N)) \text{ be } n.y \text{ in } R && \text{by (H-NAT}'\text{)} \\
&= \text{let } M \text{ be } n.x \text{ in } (\text{let } \mathrm{bind}^{\mathrm{H}}(n,N) \text{ be } z^{*n} \text{ in } (\text{let } z \text{ be } n.y \text{ in } R)^{*n}) && \text{by CC} \\
&= \text{let } M \text{ be } n.x \text{ in } (R[N/y]) && \text{by (H-}\beta\text{-}\Sigma^*\text{)}
\end{aligned}$$

It is interesting to note that a freefrom type appears in the commuting conversion step.

11.3.2 Interpretation

The only rule whose interpretation is not yet given in the above discussion is the rule (H-NAT). We sketch its interpretation here. We consider the special case of the rule where M is a variable z. The general case follows by substitution. The interpretation of the premises of (H-NAT) yields morphisms $N\colon A \to B$ and $R\colon B \to W_{\mathbf{N}}C$ in $\mathbb{E}_{\Gamma * \mathbf{N}}$. Consider the following diagram in $\mathbb{E}_{\Gamma}$.

$$\begin{array}{ccccccc}
\Sigma^*_{\mathbf{N}}A & \xrightarrow{\Sigma^*_{\mathbf{N}}N} & \Sigma^*_{\mathbf{N}}B & \xrightarrow{\Sigma^*_{\mathbf{N}}R} & \Sigma^*_{\mathbf{N}}W_{\mathbf{N}}C & \xrightarrow{\varepsilon_C} & C \\
\downarrow{\scriptstyle i} & & \downarrow{\scriptstyle i} & & & & \\
\Pi^*_{\mathbf{N}}A & \xrightarrow[\Pi^*_{\mathbf{N}}N]{} & \Pi^*_{\mathbf{N}}B & & & &
\end{array}$$

The composite of the top row is the adjoint transpose of $R \circ N\colon A \to W_{\mathbf{N}}C$, and so corresponds to the term $(\text{let } z \text{ be } n.x \text{ in } R[N/y])$. Consider the composite $\varepsilon_C \circ \Sigma^*_{\mathbf{N}}R \circ i^{-1} \circ \Pi^*_{\mathbf{N}}N \circ i$. The map $\varepsilon_C \circ \Sigma^*_{\mathbf{N}}R$ corresponds to the term $(\text{let } u \text{ be } n.y \text{ in } R)$. Since the map $\Pi^*_{\mathbf{N}}N$ corresponds to the term $\lambda^* n\colon \mathbf{N}.N[z@n/x]$, and by the definition of the terms for H-types, precomposing $\varepsilon_C \circ \Sigma^*_{\mathbf{N}}R$ with $i^{-1} \circ \Pi^*_{\mathbf{N}}N \circ i$ amounts to substituting $\lambda^{\mathrm{H}} n.N[z@_{\mathrm{H}} n/x]$ for u in $(\text{let } u \text{ be } n.y \text{ in } R)$. Hence, the composite corresponds to the term $(\text{let } (\lambda^{\mathrm{H}} n.N[z@_{\mathrm{H}} n/x]) \text{ be } n.y \text{ in } R)$. Since the diagram commutes by naturality of i, this shows the equation (H-NAT).

11.3.3 Examples

We give examples to illustrate the use of hidden-name types.

11.3.3.1 Unique choice of fresh names

In the first example, we derive a term for the unique choice of fresh names. By the choice of fresh names we mean that given a term M that makes use of a fresh name n, we can form a term new $n.M$, whose

denotation is obtained by first choosing an arbitrary fresh name for n and then taking the value of M. Since we want our type theory to be free of side-effects, we must make sure that the choice of the fresh name n has no effect, i.e. we consider *unique* choice of fresh names only. This absence of side-effect can be ensured using freefrom types. An introduction rule for unique choice of names can be formulated as:

$$\frac{\Gamma \vdash A\ \mathrm{Type} \qquad \Gamma * n\colon \mathbf{N} \vdash M : A^{*(n:\mathbf{N})}}{\Gamma \vdash \mathrm{new}\, n.M : A}$$

Essentially, this rule appears in the type system of FreshML 2000. It can be derived by the following derivation, in which we make essential use of the fact that H can be viewed both as Σ^* and as Π^*.

$$\frac{\dfrac{\dfrac{\vdots}{\Gamma * n\colon \mathbf{N} \vdash \mathrm{unit} : 1}}{\Gamma \vdash \lambda^{\mathrm{H}} n.\,\mathrm{unit} : \mathrm{H}n.\,1}\ (\text{H-I-}\Pi^*) \qquad (\textsc{Weak})\ \dfrac{\Gamma * n\colon \mathbf{N} \vdash M : A^{*(n:\mathbf{N})}}{(\Gamma * n\colon \mathbf{N}),\, u\colon 1 \vdash M : A^{*(n:\mathbf{N})}}}{\Gamma \vdash \mathrm{let}\ (\lambda^{\mathrm{H}} n.\,\mathrm{unit})\ \mathrm{be}\ n.u\ \mathrm{in}\ M : A}\ (\text{H-E-}\Sigma^*)$$

We take $(\mathrm{new}\, n.M)$ as an abbreviation for the term in the conclusion of this derivation.

Semantically, the unique choice of new names is essentially given by the counit $\varepsilon_A\colon \mathrm{H}W_{\mathbf{N}}A \to A$ of the adjunction $\mathrm{H} \dashv W_{\mathbf{N}}$, which corresponds to the term $z\colon \mathrm{H}n.\,\big(A^{*(n:\mathbf{N})}\big) \vdash \mathrm{let}\ z\ \mathrm{be}\ n.x\ \mathrm{in}\ x : A$. We shall see below that $(\mathrm{new}\, n.M)$ is provably equal to $(\mathrm{let}\ (\lambda^{\mathrm{H}} n.M)\ \mathrm{be}\ n.x\ \mathrm{in}\ x)$.

We derive some useful equations for new. The first equation states that the term $(\mathrm{let}\ M\ \mathrm{be}\ n.x\ \mathrm{in}\ N)$ can be implemented using new.

$$\frac{\Gamma \vdash M : \mathrm{H}n.A \qquad (\Gamma * n\colon \mathbf{N}),\, x\colon A \vdash N : B^{*(n:\mathbf{N})}}{\Gamma \vdash \mathrm{let}\ M\ \mathrm{be}\ n.x\ \mathrm{in}\ N = \mathrm{new}\, n.N[M@_{\mathrm{H}}n/x] : B}$$

This rule corresponds the computational intuition, as realised by $\mathbf{Ass}_S(P)$, that $(\mathrm{let}\ M\ \mathrm{be}\ n.x\ \mathrm{in}\ N)$ first instantiates the α-equivalence class M at a fresh name n, then binds the result to x and finally computes N. The rule follows directly from (H-NAT):

$$\frac{\dfrac{\dfrac{\Gamma * n\colon \mathbf{N} \vdash \mathrm{unit} : 1}{\Gamma \vdash \lambda^{\mathrm{H}} n.\,\mathrm{unit} : \mathrm{H}n.\,1} \qquad \dfrac{\dfrac{\Gamma \vdash M : \mathrm{H}n.A}{\Gamma * n\colon \mathbf{N} \vdash M@_{\mathrm{H}}n : A}}{(\Gamma * n\colon \mathbf{N}),\, u\colon 1 \vdash M@_{\mathrm{H}}n : A} \qquad (\Gamma * n\colon \mathbf{N}),\, x\colon A \vdash N : B^{*(n:\mathbf{N})}}{\Gamma \vdash \mathrm{let}\ (\lambda^{\mathrm{H}} n.\,(M@_{\mathrm{H}}n)[(\lambda^{\mathrm{H}} n.\,\mathrm{unit})@_{\mathrm{H}}n/u])\ \mathrm{be}\ n.x\ \mathrm{in}\ N = \mathrm{let}\ (\lambda^{\mathrm{H}} n.\,\mathrm{unit})\ \mathrm{be}\ n.u\ \mathrm{in}\ N[M@_{\mathrm{H}}n/x] : B}}{\Gamma \vdash \mathrm{let}\ M\ \mathrm{be}\ n.x\ \mathrm{in}\ N = \mathrm{let}\ (\lambda^{\mathrm{H}} n.\,\mathrm{unit})\ \mathrm{be}\ n.u\ \mathrm{in}\ N[M@_{\mathrm{H}}n/x] : B}$$
$$\frac{}{\Gamma \vdash \mathrm{let}\ M\ \mathrm{be}\ n.x\ \mathrm{in}\ N = \mathrm{new}\, n.N[M@_{\mathrm{H}}n/x] : B}$$

As a special case of this equation, we obtain the equality $(\mathrm{new}\, n.M) = (\mathrm{let}\ (\lambda^{\mathrm{H}} n.M)\ \mathrm{be}\ n.x\ \mathrm{in}\ x)$.

Next, we derive an equation that states that new behaves like a choice of fresh name. This means that, given M of type $A^{*(n:\mathbf{N})}$, the term $(\mathrm{new}\, n.M)$ of type A denotes the A-part of M.

$$\frac{\Gamma * n\colon \mathbf{N} \vdash M : A^{*(n:\mathbf{N})}}{\Gamma * n\colon \mathbf{N} \vdash M = (\mathrm{new}\, n.M)^{*n} : A^{*(n:\mathbf{N})}}$$

First, using (H-η-$\Sigma^*\Pi^*$), we derive:

$$\dfrac{n\colon \mathbf{N} \vdash A^{*(n:\mathbf{N})}\ \text{Type} \qquad \Gamma * n\colon \mathbf{N} \vdash n : \mathbf{N} \qquad \dfrac{\dfrac{\Gamma * n\colon \mathbf{N} \vdash M : A^{*(n:\mathbf{N})}}{\Gamma \vdash \lambda^{\mathrm{H}} n.M : \mathrm{H}n.A^{*(n:\mathbf{N})}}}{\Gamma * n\colon \mathbf{N} \vdash (\lambda^{\mathrm{H}} n.M)^{*n} : \left(\mathrm{H}n.A^{*(n:\mathbf{N})}\right)^{*(n:\mathbf{N})}}}{\Gamma * n\colon \mathbf{N} \vdash (\lambda^{\mathrm{H}} n.M)^{*n} = \mathrm{bind}^{\mathrm{H}}(n, \text{let } (\lambda^{\mathrm{H}} n.M)^{*n} \text{ be } x^{*n} \text{ in } x@_{\mathrm{H}} n) : \left(\mathrm{H}n.A^{*(n:\mathbf{N})}\right)^{*(n:\mathbf{N})}}$$

By (FF-β), (H-β-Π^*) and congruence, the right-hand side of this equation equals $\mathrm{bind}^{\mathrm{H}}(n,M)$. Using this, the required equation can be shown as follows.

$$\begin{aligned}
(\text{new } n.M)^{*n} &= (\text{let } (\lambda^{\mathrm{H}} n.M) \text{ be } n.x \text{ in } x)^{*n} && \text{special case above}\\
&= \text{let } (\lambda^{\mathrm{H}} n.M)^{*n} \text{ be } y^{*n} \text{ in } (\text{let } y \text{ be } n.x \text{ in } x)^{*n} && \text{by (FF-}\beta\text{)}\\
&= \text{let } \mathrm{bind}^{\mathrm{H}}(n,M) \text{ be } y^{*n} \text{ in } (\text{let } y \text{ be } n.x \text{ in } x)^{*n} && \text{above equation}\\
&= M && \text{by (H-}\beta\text{-}\Sigma^*\text{)}
\end{aligned}$$

It should be said that the derivation of this equation has a very simple semantic correspondence. It follows from the commuting triangle

$$\begin{array}{ccc}
W_{\mathbf{N}}\Sigma^*_{\mathbf{N}}W_{\mathbf{N}} & \xrightarrow{W_{\mathbf{N}}\varepsilon} & W_{\mathbf{N}}A \\
{\scriptstyle W_{\mathbf{N}}i_{W_{\mathbf{N}}}}\downarrow & \nearrow_{\varepsilon'_{W_{\mathbf{N}}}} & \\
W_{\mathbf{N}}\Pi^*_{\mathbf{N}}W_{\mathbf{N}} & &
\end{array}$$

in which i, ε and ε' are defined as in Section 10.1. Similar triangles commute for the other units and counits. It can be shown that if either of the two triangles in Proposition 10.1.2 commutes then so does the triangle above. In the syntax, this is reflected by the fact that the equation $M = (\text{new } n.M)^{*n}$ can be derived using the rule (H-η-$\Sigma^*\Pi^*$), as shown above, but it also follows easily using (H-β-$\Pi^*\Sigma^*$).

Another equation for new states that vacuous choice of a fresh name has no effect.

$$\dfrac{\vdash A\ \text{Type} \qquad \Gamma \vdash M : A}{\Gamma \vdash \text{new } n.\,(M^{*n}) = M : A}$$

This equation follows immediately from $(\text{new } n.\,(M^{*n}) = \text{let } (\lambda^{\mathrm{H}} n.M^{*n}) \text{ be } n.x \text{ in } x)$, which we have shown above, together with equation (H-β-$\Pi^*\Sigma^*$).

As a final equation for new, we show that the abstraction $\lambda^{\mathrm{H}} n.M$ can be implemented using new and the Σ^*-view of hidden-name types. Given $\Gamma * n\colon A \vdash M : B$, where B is a closed type, the equation

$$\lambda^{\mathrm{H}} n.M = \text{new } n.\,\mathrm{bind}^{\mathrm{H}}(n,M)$$

is derivable. Such an implementation of $\lambda^{\mathrm{H}} n.M$ has been used by Gabbay, who in [36] frequently uses $(\text{new } n.\,\mathrm{bind}^{\mathrm{H}}(n,M))$ to the effect of a lambda-abstraction. But note that $\lambda^{\mathrm{H}} n.M$ is simpler than

$(\text{new } n.\,\text{bind}^{\text{H}}(n,M))$, since it does not require a proof that the choice of the fresh name has no effect. The equation can be shown as follows:

$$\begin{aligned}
\lambda^{\text{H}} n.M &= \text{let } (\lambda^{\text{H}} n.M) \text{ be } n.x \text{ in } (\text{let bind}^{\text{H}}(n,x) \text{ be } z^{*n} \text{ in } z^{*n}) && \text{by (H-}\eta\text{-}\Sigma^*\text{)}\\
&= \text{let } (\lambda^{\text{H}} n.M) \text{ be } n.x \text{ in bind}^{\text{H}}(n,x) && \text{by (FF-}\eta\text{)}\\
&= \text{new } n.\,\text{bind}^{\text{H}}(n,M) && \text{see above}
\end{aligned}$$

11.3.3.2 Extensionality of hidden-name types

Because of the isomorphism $\Sigma^*_{\mathbf{N}} \to \Pi^*_{\mathbf{N}}$, on which hidden-name types are based, the functions of type $\Pi^* n\colon \mathbf{N}.A$ are completely determined by their value at a single fresh name. Making essential use of the interaction equations for hidden-name types, we show functional extensionality of $\text{H}n.A$, and thus also of $\Pi^* n\colon \mathbf{N}.A$. We construct an equality proof p as in the following sequent, in which $\simeq_B$ denotes a standard intensional identity type, see e.g. [47].

$$(f\colon \text{H}n.A, g\colon \text{H}n.A) * m\colon \mathbf{N}, e\colon (f@_{\text{H}}m) \simeq_{A[m/n]} (g@_{\text{H}}m) \vdash p : f \simeq_{\text{H}n.A} g$$

The proof p is constructed by the following calculation.

$$\begin{aligned}
f &\simeq_{(\text{H}n.A)} \text{let } f^{*m} \text{ be } x^{*m} \text{ in } x && \text{by (FF-}\beta\text{)}\\
&\simeq_{(\text{H}n.A)} \text{let bind}^{\text{H}}(m, \text{let } f^{*m} \text{ be } y^{*m} \text{ in } y@_{\text{H}}m) \text{ be } x^{*m} \text{ in } x && \text{by (H-}\eta\text{-}\Sigma^*\Pi^*\text{)}\\
&\simeq_{(\text{H}n.A)} \text{let bind}^{\text{H}}(m, f@_{\text{H}}m) \text{ be } x^{*m} \text{ in } x && \text{by (FF-}\beta\text{)}\\
&\simeq_{(\text{H}n.A)} \text{let bind}^{\text{H}}(m, g@_{\text{H}}m) \text{ be } x^{*m} \text{ in } x && \text{using assumption } e\\
&\simeq_{(\text{H}n.A)} g && \text{as for } f
\end{aligned}$$

Except for the step in which assumption e is used, these equations are in fact definitional equations.

A direct consequence of the above sequent is the following principle of functional extensionality.

$$(f\colon \text{H}n.A, g\colon \text{H}n.A, h\colon \text{H}n.\,(f@_{\text{H}}n) \simeq_A (g@_{\text{H}}n)) * m\colon \mathbf{N} \vdash q : f \simeq_{(\text{H}n.A)} g$$

Even though the name m is not mentioned in the types of this sequent, it is used essentially in the term q. The obvious way of getting a sequent that does not mention the name m is to use the unique choice of fresh names to choose m freshly. But to do so we would have to show that q has type $(f \simeq_{(\text{H}n.A)} g)^{*(m:\mathbf{N})}$, and this is an open freefrom type.

11.3.3.3 Name-replacement and swapping

Another example of what we can do by mixing the two views of H as Σ^* and Π^* is a generic name replacement. Suppose we have a term M of closed type A and a name n and we want to replace the name n by a fresh name m in the value denoted by M. This name replacement can be implemented by the term $(n.M)@m$. In words, we first bind the name n in M and then instantiate the result at the fresh

name m. The intuition for binding and instantiation suggest that $(n.M)@m$ denotes the term M with n replaced by m. Formally, however, hidden-name types are just syntax for the structure of a category with bindable names. So, is it justified to think of $(n.M)@m$ as M with n replaced by m? In this section we argue that this is the case, by giving examples that indicate that $(n.M)@m$ behaves like such a replacement.

We start by looking at the case where M is a name n, and derive the equation $(n.n)@m = m$. Let Γ be a context and assume a name $\Gamma \vdash n : \mathbf{N}$. In context Γ we can derive the following equations of terms of type $(\mathrm{H}n.\mathbf{N})^{*(n:\mathbf{N})}$.

$$\begin{aligned}
\mathrm{bind}(n,n) &= \mathrm{bind}(n,(\lambda^{\mathrm{H}}m.m)@_{\mathrm{H}}n) && \text{by (H-}\beta\text{-}\Pi^*\text{)}\\
&= \mathrm{bind}(n,\text{let } (\lambda^{\mathrm{H}}m.m)^{*n} \text{ be } x^{*n} \text{ in } x@_{\mathrm{H}}n) && \text{by (FF-}\beta\text{)}\\
&= (\lambda^{\mathrm{H}}m.m)^{*n} && \text{by (H-}\eta\text{-}\Sigma^*\Pi^*\text{)}
\end{aligned}$$

Since $(n.n) = (\lambda^{\mathrm{H}}m.m)$ follows by definition of the abbreviation $M.N$, we have just shown the rule

$$\frac{\Gamma \vdash n : \mathbf{N}}{\Gamma \vdash n.n = \lambda^{\mathrm{H}}m.m : \mathrm{H}n.\mathbf{N}}$$

The required equality $(n.n)@_{\mathrm{H}}m = m$ is now a direct consequence of this rule. Another consequence is that we can show $(n.n) = (m.m)$ for any two names $\Gamma \vdash n : \mathbf{N}$ and $\Gamma \vdash m : \mathbf{N}$.

By almost exactly the same argument, we can also derive the equation

$$n\colon \mathbf{N} * m\colon \mathbf{N} \vdash n.m = \lambda^{\mathrm{H}}n.m : \mathrm{H}n.\mathbf{N}.$$

This shows that, for different names n and m, we have $(n.m)@r = m$.

Having shown that, on names, $(n.M)@m$ behaves like a replacement operation, we give an example to show that if M is non-primitive then it is possible to move the renaming operation inside the subterms. We consider the case where M is a pair $\langle x,y\rangle$, and we show the equation $(n.\langle x,y\rangle)@m = \langle (n.x)@m, (n.y)@m\rangle$, which would be expected of a name-replacement. We begin by deriving the following rule.

$$\frac{\vdash A \text{ Type} \qquad \vdash B \text{ Type} \qquad \Gamma \vdash n : \mathbf{N} \qquad \Gamma \vdash x : A \qquad \Gamma \vdash y : B}{\Gamma \vdash n.\langle x,y\rangle = \lambda^{\mathrm{H}}m.\,\langle (n.x)@_{\mathrm{H}}m, (n.y)@_{\mathrm{H}}m\rangle : \mathrm{H}n.\,(A \times B)}$$

To this end, we first show that, under the assumptions of the rule, the following equation is derivable.

$$\Gamma \vdash \mathrm{bind}(n,\langle x,y\rangle) = \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } (\lambda^{\mathrm{H}}m.\,\langle u@_{\mathrm{H}}m, v@_{\mathrm{H}}m\rangle)^{*n} : (\mathrm{H}n.A \times B)^{*(n:\mathbf{N})}$$

The term J used in this sequent is defined below.

$$J \stackrel{\text{def}}{=} \mathrm{join}(\mathrm{bind}(n,x), \mathrm{bind}(n,y))$$

We show this equation by the following calculation in context Γ.

$$
\begin{aligned}
&\mathrm{bind}(n, \langle x, y\rangle) \\
&= \mathrm{bind}(n, \langle \text{let } \mathrm{bind}(n,x) \text{ be } u^{*n} \text{ in } u@_{\mathrm{H}}n, \text{let } \mathrm{bind}(n,y) \text{ be } v^{*n} \text{ in } v@_{\mathrm{H}}n\rangle) && \text{by (H-}\beta\text{-}\Sigma^*\Pi^*\text{)} \\
&= \mathrm{bind}(n, \langle \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } u@_{\mathrm{H}}n, \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } v@_{\mathrm{H}}n\rangle) && \text{see below} \\
&= \mathrm{bind}(n, \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } \langle u@_{\mathrm{H}}n, v@_{\mathrm{H}}n\rangle) && \text{by CC} \\
&= \mathrm{bind}(\text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } n, \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } \langle u@_{\mathrm{H}}n, v@_{\mathrm{H}}n\rangle) && \text{by (FF-}\eta\text{)} \\
&= \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } \mathrm{bind}(n, \langle u@_{\mathrm{H}}n, v@_{\mathrm{H}}n\rangle) && \text{by CC} \\
&= \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } \mathrm{bind}(n, (\lambda^{\mathrm{H}}m.\, \langle u@_{\mathrm{H}}m, v@_{\mathrm{H}}m\rangle)@_{\mathrm{H}}n) && \text{by (H-}\beta\text{-}\Pi^*\text{)} \\
&= \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } \mathrm{bind}(n, \text{let } (\lambda^{\mathrm{H}}m.\, \langle u@_{\mathrm{H}}m, v@_{\mathrm{H}}m\rangle)^{*n} \text{ be } z^{*n} \text{ in } z@_{\mathrm{H}}n) && \text{by (FF-}\beta\text{)} \\
&= \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } (\lambda^{\mathrm{H}}m.\, \langle u@_{\mathrm{H}}m, v@_{\mathrm{H}}m\rangle)^{*n} && \text{by (H-}\eta\text{-}\Sigma^*\Pi^*\text{)}
\end{aligned}
$$

The second step in this chain is valid because (FF-JOIN-EQ1) gives us the equation

$$\Gamma \vdash (\text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } u^{*n}) = \mathrm{bind}(n,x) : (\mathrm{H}n.A)^{*(n:\mathbf{N})},$$

using which the we have

$$
\begin{aligned}
&\text{let } \mathrm{bind}(n,x) \text{ be } u^{*n} \text{ in } u@_{\mathrm{H}}n \\
&= \text{let } (\text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } u^{*n}) \text{ be } u^{*n} \text{ in } u@_{\mathrm{H}}n && \text{by above eqn.} \\
&= \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } (\text{let } u^{*n} \text{ be } u^{*n} \text{ in } u@_{\mathrm{H}}n) && \text{by CC} \\
&= \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } u@_{\mathrm{H}}n && \text{by (FF-}\beta\text{)}
\end{aligned}
$$

We have thus shown the equation

$$\Gamma \vdash \mathrm{bind}(n, \langle x,y\rangle) = \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } (\lambda^{\mathrm{H}}m.\, \langle u@_{\mathrm{H}}m, v@_{\mathrm{H}}m\rangle)^{*n} : (\mathrm{H}n.A \times B)^{*(n:\mathbf{N})}.$$

The rule that we set out to show is a consequence of this equation:

$$
\begin{aligned}
n.\langle x,y\rangle &= \text{let } \mathrm{bind}(n, \langle x,y\rangle) \text{ be } z^{*n} \text{ in } z && \text{by defn.} \\
&= \text{let } \big(\text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } (\lambda^{\mathrm{H}}m.\, \langle u@_{\mathrm{H}}m, v@_{\mathrm{H}}m\rangle)^{*n}\big) \text{ be } z^{*n} \text{ in } z && \text{by above eqn.} \\
&= \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } (\text{let } (\lambda^{\mathrm{H}}m.\, \langle u@_{\mathrm{H}}m, v@_{\mathrm{H}}m\rangle)^{*n} \text{ be } z^{*n} \text{ in } z) && \text{by CC} \\
&= \text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } (\lambda^{\mathrm{H}}m.\, \langle u@_{\mathrm{H}}m, v@_{\mathrm{H}}m\rangle) && \text{by (FF-}\beta\text{)} \\
&= \lambda^{\mathrm{H}}m.\, \langle (\text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } u)@_{\mathrm{H}}m, (\text{let } J \text{ be } \langle u,v\rangle^{*n} \text{ in } v)@_{\mathrm{H}}m\rangle && \text{by CC} \\
&= \lambda^{\mathrm{H}}m.\, \langle (\text{let } \mathrm{bind}(n,x) \text{ be } u^{*n} \text{ in } u)@_{\mathrm{H}}m, (\text{let } \mathrm{bind}(n,y) \text{ be } v^{*n} \text{ in } v)@_{\mathrm{H}}m\rangle && \text{as above} \\
&= \lambda^{\mathrm{H}}m.\, \langle (n.x)@_{\mathrm{H}}m, (n.y)@_{\mathrm{H}}m\rangle && \text{by defn.}
\end{aligned}
$$

The above examples make it reasonable to expect that the operation taking M to $(n.M)@m$ gives rise to a kind of name-replacement operation in any category with bindable names. Due to the restriction to

closed freefrom types, however, it is likely that not all equations that we expect to hold for $(n.M)@m$ are derivable. Furthermore, the above examples show that even derivations of simple equations can be quite tedious.

Gabbay and Pitts have argued [38, 86] that name-swapping enjoys better properties than name-replacement. They argue that it is better to consider name-swapping $(m\ n)\cdot M$, which can be formed regardless of whether or not m and n are fresh for M, as opposed to $(n.M)@m$, which can be formed only if m is fresh for $n.M$. As should be expected, such a swapping $(m\ n)\cdot M$ can be derived using name-replacements (Shinwell's FreshML 2000 used to contain such a definition as an example). In the rest of this section, we define such a swapping function for closed types A.

$$n\colon \mathbf{N}, m\colon \mathbf{N}, x\colon A \vdash \mathrm{swap}(n,m,x) : A$$

We start by comparing the names n and m. If they are equal then swapping should have no effect, so we let $\mathrm{swap}(n,m,x)$ be x. If they are different, then it suffices to define a term S as in the following sequent.

$$(n\colon \mathbf{N} * m\colon \mathbf{N}), x\colon A \vdash S : A$$

Informally, the term S can be described as "new $r.(r.(n.((m.x)@r)@m))@n$". Notice that from this informal description it is not immediately obvious that all the uses of $@$ are valid, i.e. that the names that are applied are provably fresh for the respective bodies. A proof of this freshness information will be contained in the term S, whose derivation we start as follows:

$$\dfrac{\dfrac{(n\colon \mathbf{N} * m\colon \mathbf{N} * r\colon \mathbf{N}), x\colon A^{*(r:\mathbf{N})} \vdash S' : A^{*(r:\mathbf{N})}}{((n\colon \mathbf{N} * m\colon \mathbf{N}), x\colon A) * r\colon \mathbf{N} \vdash S'[x^{*r}/x] : A^{*(r:\mathbf{N})}}}{(n\colon \mathbf{N} * m\colon \mathbf{N}), x\colon A \vdash S \stackrel{\mathrm{def}}{=} \mathrm{new}\ r.S'[x^{*r}/x] : A}$$

For the definition of S' we define two terms:

$$(n\colon \mathbf{N} * m\colon \mathbf{N} * r\colon \mathbf{N}), x\colon (\mathrm{H}r.A)^{*(m*r:\mathbf{N}*\mathbf{N})} \vdash \text{“}x@r\text{”} : A^{*(m:\mathbf{N})}$$
$$(n\colon \mathbf{N} * m\colon \mathbf{N} * r\colon \mathbf{N}), x\colon A^{*(r:\mathbf{N})} \vdash \text{“}m.x\text{”} : (\mathrm{H}m.A)^{*(m*r:\mathbf{N}*\mathbf{N})}$$

These terms intuitively amount to application $(x@r)$ and binding $(m.x)$ but in order for them to have the types shown in the above sequents, they need to contain book-keeping constructs that rearrange the freshness information appropriately. The term "$x@r$" is defined by the judgement

$$(n\colon \mathbf{N} * m\colon \mathbf{N} * r\colon \mathbf{N}), x\colon (\mathrm{H}r.A)^{*(m*r:\mathbf{N}*\mathbf{N})} \vdash \mathrm{let}\ x\ \mathrm{be}\ z^{*p}\ \mathrm{in}\ (\mathrm{let}\ p\ \mathrm{be}\ u*v\ \mathrm{in}\ (z@_{\mathrm{H}}v)^{*u}) : A^{*(m:\mathbf{N})},$$

which is easily seen to be derivable. The term "$m.x$" is defined as the term in the conclusion of the following derivation.

$$\dfrac{\dfrac{(n\colon \mathbf{N} * m\colon \mathbf{N}), x\colon A \vdash \mathrm{bind}(m,x) : (\mathrm{H}m.A)^{*(m:\mathbf{N})} \qquad (m\colon \mathbf{N} * z\colon \mathrm{H}m.A) * r\colon \mathbf{N} \vdash z^{*m*r} : (\mathrm{H}m.A)^{*(m*r:\mathbf{N}*\mathbf{N})}}{((n\colon \mathbf{N} * m\colon \mathbf{N}), x\colon A) * r\colon \mathbf{N} \vdash \mathrm{let}\ \mathrm{bind}(m,x)\ \mathrm{be}\ z^{*m}\ \mathrm{in}\ z^{*m*r} : (\mathrm{H}m.A)^{*(m*r:\mathbf{N}*\mathbf{N})}}}{(n\colon \mathbf{N} * m\colon \mathbf{N} * r\colon \mathbf{N}), x\colon A^{*(r:\mathbf{N})} \vdash \mathrm{let}\ x\ \mathrm{be}\ x^{*r}\ \mathrm{in}\ (\mathrm{let}\ \mathrm{bind}(m,x)\ \mathrm{be}\ z^{*m}\ \mathrm{in}\ z^{*m*r}) : (\mathrm{H}m.A)^{*(m*r:\mathbf{N}*\mathbf{N})}}$$

We are slightly imprecise in the last step of this derivation. In this step we use the elimination of open freefrom types for notational convenience. Making use of the internalisation of contexts and join, this step can be simulated by (a more complicated) derivation using only closed freefrom types.

Using the defined terms "$x@r$" and "$m.x$" we can define S'. Write Γ as an abbreviation for the context $(n\colon \mathbf{N} * m\colon \mathbf{N} * r\colon \mathbf{N}), x\colon A^{*(r:\mathbf{N})}$. The term that amounts to $(r.(n.((m.x)@r)@m))@n$ is now defined by successive substitution.

$$\Gamma \vdash \text{“}m.x\text{”} : (\mathrm{H}m.A)^{*(m*r:\mathbf{N}*\mathbf{N})}$$
$$\Gamma \vdash \text{“}x@r\text{”}[\text{“}m.x\text{”}/x] : A^{*(m:\mathbf{N})}$$
$$\Gamma \vdash \text{“}n.x\text{”}[\text{“}x@r\text{”}[\text{“}m.x\text{”}/x]/x] : (\mathrm{H}n.A)^{*(n*m:\mathbf{N}*\mathbf{N})}$$
$$\Gamma \vdash \text{“}x@m\text{”}[\text{“}n.x\text{”}[\text{“}x@r\text{”}[\text{“}m.x\text{”}/x]/x]/x] : A^{*(n:\mathbf{N})}$$
$$\Gamma \vdash \text{“}r.x\text{”}[\text{“}x@m\text{”}[\text{“}n.x\text{”}[\text{“}x@r\text{”}[\text{“}m.x\text{”}/x]/x]/x]/x] : (\mathrm{H}r.A)^{*(r*n:\mathbf{N}*\mathbf{N})}$$
$$\Gamma \vdash \text{“}x@n\text{”}[\text{“}r.x\text{”}[\text{“}x@m\text{”}[\text{“}n.x\text{”}[\text{“}x@r\text{”}[\text{“}m.x\text{”}/x]/x]/x]/x]/x] : A^{*(r:\mathbf{N})}$$

We let S' be the term in the last sequent.

It is straightforward to show that in the Schanuel topos and in the realizability category $\mathbf{Ass}_S(P)$ the interpretation of swap is indeed the swapping function. We conjecture that, in any category with bindable names, swap satisfies equations that make it a swapping action, cf. Section 10.5.

11.3.3.4 Freshness for hidden-name types

We end the section on hidden-name types with an example that we can only derive using open freefrom types. We define a term M of the following type

$$n\colon \mathbf{N}, x\colon \mathrm{H}m.\left(A^{*(n:\mathbf{N})}\right) \vdash M : \left(\mathrm{H}m.A\right)^{*(n:\mathbf{N})},$$

where A is a closed type. Informally, the term expresses that in order for n to be fresh for y of type $\mathrm{H}m.A$, it suffices for n to be fresh for all the values $y@m$. For abstract syntax, this corresponds to the fact that a name is fresh for an α-equivalence class if it is fresh for all freshly named instances of that α-equivalence class. Having such a term M is useful for defining elements of freefrom types by structural recursion, see Section 12.1.3 for an example.

Recall from Section 8.2 that open freefrom types can be used to derive a term $N_B(z)$ that fits in the rule below.

$$\frac{\vdash B\ \mathrm{Type}}{m\colon \mathbf{N}, n\colon \mathbf{N}, z\colon \left(B^{*(m:\mathbf{N})}\right)^{*(n:\mathbf{N})} \vdash N_B(z) : \left(B^{*(n:\mathbf{N})}\right)^{*(m:\mathbf{N})}}$$

We use this term in the following derivation, in which we write R for $N_{(\mathrm{H}m.A)}(\mathrm{bind}^{\mathrm{H}}(m,x)^{*n})$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(m\colon \mathbf{N}, x\colon A) * n\colon \mathbf{N} \vdash \mathrm{bind}^{\mathrm{H}}(m,x)^{*n} : \left((\mathrm{H}m.A)^{*(m:\mathbf{N})}\right)^{*(n:\mathbf{N})}}{(m\colon \mathbf{N}, x\colon A) * n\colon \mathbf{N} \vdash R : \left((\mathrm{H}m.A)^{*(n:\mathbf{N})}\right)^{*(m:\mathbf{N})}}}{(m\colon \mathbf{N} * n\colon \mathbf{N}), y\colon A^{*(n:\mathbf{N})} \vdash \text{let } y \text{ be } x^{*n} \text{ in } R : \left((\mathrm{H}m.A)^{*(n:\mathbf{N})}\right)^{*(m:\mathbf{N})}}}{(n\colon \mathbf{N} * m\colon \mathbf{N}), y\colon A^{*(n:\mathbf{N})} \vdash \text{let } y \text{ be } x^{*n} \text{ in } R : \left((\mathrm{H}m.A)^{*(n:\mathbf{N})}\right)^{*(m:\mathbf{N})}}}{n\colon \mathbf{N}, z\colon \mathrm{H}m.\left(A^{*(n:\mathbf{N})}\right) \vdash \text{let } z \text{ be } m.y \text{ in let } y \text{ be } x^{*n} \text{ in } R : (\mathrm{H}m.A)^{*(n:\mathbf{N})}}$$

Informally, the above derivation amounts to the following reasoning. An element of $\mathrm{H}m.\left(A^{*(n:\mathbf{N})}\right)$ can be viewed as a pair $m.y$, where m is a fresh name and y is an element of $A^{*(n:\mathbf{N})}$. From y we get an element x of type A that is fresh for n. Now consider $m.x$. Since both m and x are fresh for n, it must be that case that $m.x$ is also fresh for n, i.e. $m.x$ is an element of type $(\mathrm{H}m.A)^{*(n:\mathbf{N})}$. Moreover, m is fresh for $m.x$, so that the choice of the original fresh name m does not matter.

In the above derivation, we have used only the Σ^*-view of hidden-name types. Open freefrom types are needed to use the Σ^*-elimination rule and to derive $N_{\mathrm{H}m.A}(z)$. We do not know if it is possible to give a corresponding derivation using only the Π^*-view of hidden-name types.

Going from $(\mathrm{H}m.A)^{*(n:\mathbf{N})}$ to $\mathrm{H}m.\left(A^{*(n:\mathbf{N})}\right)$, on the other hand, is possible using only the Π^*-view of hidden-name types. This direction is given by the following term, which furthermore can be derived using only closed freefrom types.

$$n\colon \mathbf{N}, x\colon (\mathrm{H}m.A)^{*(n:\mathbf{N})} \vdash \text{let } x \text{ be } y^{*n} \text{ in } \lambda^{\mathrm{H}}m.\,(y@_{\mathrm{H}}m)^{*n} : \mathrm{H}m.\left(A^{*(n:\mathbf{N})}\right)$$

11.4 Swapping

Categories with bindable names, from which the all our type constructs for names and binding are derived, are based on freshness, i.e. the monoidal structure $*$. In contrast, in FM set theory and related work the central primitive is name-swapping. In this section we discuss informally how swapping fits into the type theory.

Even though in Section 11.3.3.3 we have given some evidence that categories with bindable names contain a name-swapping operation, at present we do not know whether the general swapping operation $(m\ n)\cdot M$ is modelled by any category with bindable names (although we conjecture that it is). Nevertheless, the intended models from Chapter 3 are all constructed using permutation actions and so contain a swapping operation.

Swapping can be added to the type theory as a special restricted kind of explicit substitution, as is done in [112] and [16].

$$(\textsc{Swap-Ty})\ \frac{\Gamma \vdash m : \mathbf{N} \qquad \Gamma \vdash n : \mathbf{N} \qquad \Gamma \vdash A\ \mathrm{Type}}{\Gamma \vdash (m\ n)\cdot A\ \mathrm{Type}}$$

$$(\textsc{Swap-Tm})\ \frac{\Gamma \vdash m : \mathbf{N} \qquad \Gamma \vdash n : \mathbf{N} \qquad \Gamma \vdash R : A}{\Gamma \vdash (m\ n)\cdot R : (m\ n)\cdot A}$$

Bunches are useful for the formulation of the equations for swapping. For instance, bunches are used in the following two equations, which state that swapping two fresh names has no effect.

$$(\textsc{Sw-Ty-Fresh})\ \frac{\Gamma \vdash A\ \mathrm{Type} \qquad \Delta \vdash m : \mathbf{N} \qquad \Delta \vdash n : \mathbf{N}}{\Gamma * \Delta \vdash (m\ n) \cdot A = A\ \mathrm{Type}}$$

$$(\textsc{Sw-Tm-Fresh})\ \frac{\Gamma \vdash M : A \qquad \Delta \vdash m : \mathbf{N} \qquad \Delta \vdash n : \mathbf{N}}{\Gamma * \Delta \vdash (m\ n) \cdot M = M : A}$$

On the other hand, bunches also complicate the formulation of the other equations for swapping. In addition to equations for swapping on names, the reader may expect equations such as $(m\ n) \cdot \langle M, N \rangle = \langle (m\ n) \cdot M, (m\ n) \cdot N \rangle$ that amount to pushing swapping inside the term structure. Such equations work well for the additive types, but they do not work for the multiplicative types. For example, for $*$-types one may expect the equation $(m\ n) \cdot (M * N) = ((m\ n) \cdot M) * ((m\ n) \cdot N)$. We cannot take this equation, however, because the term on the right-hand side is not typeable. This can be seen by noting that $M * N$ is typeable only if M and N do not share free variables. Nevertheless, it is possible to give equations for swapping for the multiplicative types. For example, one equation for $*$-types could be:

$$\frac{\vdash A, B\ \mathrm{Type} \qquad \Gamma \vdash M : A \qquad \Delta \vdash N : B \qquad \Gamma \vdash m, n : \mathbf{N}}{\Gamma * \Delta \vdash (m\ n) \cdot (M * N) = ((m\ n) \cdot M) * N : A * B}$$

Although such equations are not very easy to work with, a formulation of them is at least possible.

Other issues with the formulation of swapping include the interaction of swapping and freshness. For instance, we would expect the equation $\lambda^* r\colon \mathbf{N}.\, \lambda^* m\colon \mathbf{N}.\, (r\ n) \cdot (m\ n) \cdot x = \lambda^* r\colon \mathbf{N}.\, \lambda^* m\colon \mathbf{N}.\, (m\ n) \cdot x$ to be derivable. Intuitively, this equation should hold because swapping the name n with a fresh name m in some term x gives a term $(m\ n) \cdot x$ that is fresh for n, and so swapping n with the fresh name r in $(m\ n) \cdot x$ should not have any effect. However, we cannot immediately use the rule (Sw-Tm-Fresh) to derive this equation, since we cannot satisfy its freshness assumptions. One possibility of nevertheless obtaining the equation is to relate swapping to hidden-name types. For all $M\colon \mathrm{H}n.A$, we can add the equation $(n.M)@m = (m\ n) \cdot M$. Since, by the formulation of hidden-name types, we do know that n is fresh for $(n.M)$, we can use this equation to obtain that n is fresh for $(m\ n) \cdot M$ whenever m is fresh. This allows us to derive the desired equation.

Swapping nevertheless has some interesting applications. For instance, we can derive the equation $(x\colon A, n\colon \mathbf{N}) * m\colon \mathbf{N} \vdash n.x = m.((m\ n) \cdot x) : \mathrm{H}n.A$, which further justifies the use of $\mathrm{H}n.A$ as a representation of α-equivalence classes. The derivation makes essential use of the equation $(n.M)@m = (m\ n) \cdot M$.

Another example of what can be done with swapping is that swapping can be used to define certain open freefrom types. The definition of freefrom types from swapping is based on the idea from FM set theory that a name n is fresh for a term x if and only if $(m\ n) \cdot x = x$ holds for all fresh names m. This can be captured by the following definition.

$$\frac{\Gamma \vdash n : \mathbf{N} \qquad \Gamma * m\colon \mathbf{N} \vdash (m\ n) \cdot A = A\ \mathrm{Type}}{\Gamma \vdash A^{*(n:\mathbf{N})} \stackrel{\mathrm{def}}{=} (\Sigma x\colon A.\, \Pi^* m\colon \mathbf{N}.\, (m\ n) \cdot x \simeq_A x)\ \mathrm{Type}}$$

The assumption $\Gamma * m\colon \mathbf{N} \vdash (m\ n)\cdot A = A$ Type is needed to ensure that both x and $(m\ n)\cdot x$ have type A, as is required for $(m\ n)\cdot x \simeq_A x$ to be a type. This assumption could be lifted if we were to use, for instance, John Major (JM) equality [66] instead of standard identity types, since with JM equality it is possible to state equations between terms of different type.

With the definition of freefrom types from swapping, interesting terms can be defined. For instance, it is possible to derive the rules (FF-JOIN), (FF-JOIN-EQ1) and (FF-JOIN-EQ2) that we had to assume for closed freefrom types. If we further assume the following principle of uniqueness of identity proofs

$$\text{(UIP)}\ \frac{\Gamma \vdash p : a \simeq_A b \qquad \Gamma \vdash q : a \simeq_A b}{\Gamma \vdash \mathrm{UIP}(p,q) : p \simeq_{(a \simeq_A b)} q},$$

which is discussed in depth in [50], then more interesting examples can be derived. For instance, we can strengthen the functional extensionality for H-types from Section 11.3.3.2 and derive a term

$$f\colon \mathrm{H}n.A, g\colon \mathrm{H}n.A, h\colon \mathrm{H}n.\,(f @_{\mathrm{H}} n) \simeq_A (g @_{\mathrm{H}} n) \vdash M : f \simeq_{(\mathrm{H}n.A)} g.$$

Using this term, it is then possible to obtain a term

$$n\colon \mathbf{N}, x\colon \mathrm{H}m.\,(A^{*(n:\mathbf{N})}) \vdash N : (\mathrm{H}m.A)^{*(n:\mathbf{N})},$$

see also Section 11.3.3.4.

However, the definition of freefrom types from swapping also has limitations. For instance, to define an isomorphism

$$n\colon \mathbf{N}, m\colon \mathbf{N} \vdash \left(A^{*(n:\mathbf{N})}\right)^{*(m:\mathbf{N})} \cong \left(A^{*(m:\mathbf{N})}\right)^{*(n:\mathbf{N})} \text{ Type},$$

it appears to be necessary to assume the identity types $\simeq$ to be extensional. To see this note that an element x of type $\left(A^{*(n:\mathbf{N})}\right)^{*(m:\mathbf{N})}$ consists of an element $x'\colon A$ together with proofs

$$\begin{aligned}
&e_1 : \Pi^* r\colon \mathbf{N}.\,(r\ n)\cdot x' \simeq_A x',\\
&e_2 : \Pi^* r\colon \mathbf{N}.\,(r\ m)\cdot x' \simeq_A x',\\
&e_3 : \Pi^* r\colon \mathbf{N}.\,(r\ m)\cdot e_1 \simeq_{(\Pi^* r\colon \mathbf{N}.\,(r\ n)\cdot x' \simeq_A x')} e_1.
\end{aligned}$$

We want to give an element of $\left(A^{*(m:\mathbf{N})}\right)^{*(n:\mathbf{N})}$ whose A-part is x'. For this, we have to give proofs

$$\begin{aligned}
&f_1 : \Pi^* r\colon \mathbf{N}.\,(r\ m)\cdot x' \simeq_A x',\\
&f_2 : \Pi^* r\colon \mathbf{N}.\,(r\ n)\cdot x' \simeq_A x',\\
&f_3 : \Pi^* r\colon \mathbf{N}.\,(r\ n)\cdot f_1 \simeq_{(\Pi^* r\colon \mathbf{N}.\,(r\ m)\cdot x' \simeq_A x')} f_1.
\end{aligned}$$

The first two inhabitants f_1 and f_2 follow immediately from the assumptions. In order to give an element f_3, however, the two subterms $(r\ n)\cdot f_1$ and f_1 must have definitionally equal types, since propositional equality (and also JM equality) can only be introduced for elements of the same type. We therefore have to show that the types $((r\ n)\cdot \Pi^* r\colon \mathbf{N}.\,(r\ m)\cdot x' \simeq_A x')$ and $\Pi^* r\colon \mathbf{N}.\,(r\ m)\cdot x' \simeq_A x'$ are equal *definitionally*. The first type is provably equal to $\Pi^* r\colon \mathbf{N}.\,(r\ n)\cdot(r\ m)\cdot x' \simeq_A (r\ n)\cdot x'$. The assumptions e_1, e_2

and e_3 are enough to show that the terms $(r\,m)\cdot x'$, $(r\,n)\cdot(r\,m)\cdot x'$ and $(r\,n)\cdot x'$ are all propositionally equal to x'. But we do not get definitional equality, and there appears to be no reason why the types of $(r\,n)\cdot f_1$ and f_1 should be equal. If, on the other hand, we assume the identity types to be extensional, i.e. if we assume that propositional equality implies definitional equality, then the equality of the types of $(r\,n)\cdot f_1$ and f_1 follows at once, and we can define the term f_3 easily.

In conclusion, while rules for swapping can be added to the type theory, the presence of multiplicative types makes their formulation a good deal more complicated than perhaps expected. The examples for swapping, while interesting, also demonstrate that one is quickly lead to making strong assumptions on the type theory, such as extensional identity types.

11.5 Related Work

Type systems with names and binding have been proposed by Pitts & Gabbay [87] and Cardelli, Gardner & Ghelli [16]. Pitts & Gabbay [87] introduce FreshML 2000 with a type system that contains names and abstraction sets. This type system is related to our type system with hidden-name types, since abstraction sets and the operations on it are, semantically, a special case of hidden-name types. However, the type systems are not directly comparable, due to the different handling of freshness assumptions. Also, since Pitts & Gabbay's aim is to define a type system for a ML-like programming language, they consider the evaluation of programs but not the equational theory of the type system.

In [16] Cardelli et al. propose a language for the manipulation of trees with hidden names and give a type system for it. They introduce types of the form $\mathrm{H}n.A$ with rules that correspond roughly to our rules for Π^*-types. They also use a form of bunched contexts and integrate swapping in their type system. The work of Cardelli et al. has directly influenced our work, in particular regarding the use of bunches. However, we do not know if there is a precise formal relation between the two systems. For example, Cardelli et al. represent names as constants while we work with names as variables. Furthermore, the type system of Cardelli et al. is formulated in a different setting, closer in spirit to type systems for the π-calculus, e.g. [118], than to our system.

Chapter 12

Examples

Having added hidden-name types to the type theory, in this chapter we give some simple examples of their use for abstract syntax with variable binders. Using the untyped λ-calculus as a running example, we first define functions such as substitution and the function returning the free variables of a term. We then encode a typing judgement for the simply-typed λ-calculus and implement the evaluation of well-typed terms.

Throughout this chapter we will make extensive use of inductive types. Since we have not justified the use of inductive types in general, we shall justify their use in this chapter on a case-by-case basis.

12.1 Untyped λ-calculus

We define the syntax of the untyped λ-calculus inductively. The inductive type Lam of λ-terms and its constructors are given by the following constants. The inductive type Lam of λ-terms and its constructors are given by the following constants.

$$\vdash \mathsf{Lam}\ \mathrm{Type}$$
$$\vdash \mathsf{var} : \mathbf{N} \to \mathsf{Lam}$$
$$\vdash \mathsf{app} : \mathsf{Lam} \to \mathsf{Lam} \to \mathsf{Lam}$$
$$\vdash \mathsf{lam} : (\mathrm{H}n.\,\mathsf{Lam}) \to \mathsf{Lam}$$

The associated recursion principle is:

$$(\mathsf{Lam\text{-}REC})\ \frac{\begin{array}{l}\Gamma, x\colon \mathsf{Lam} \vdash A\ \mathrm{Type}\\ \Gamma \vdash M : \Pi n\colon \mathbf{N}.\, A[\mathsf{var}\ n/x]\\ \Gamma \vdash N : \Pi s\colon \mathsf{Lam}.\,\Pi t\colon \mathsf{Lam}.\, A[s/x] \to A[t/x] \to A[\mathsf{app}\ s\ t/x]\\ \Gamma \vdash R : \Pi s\colon (\mathrm{H}n.\,\mathsf{Lam}).\,(\mathrm{H}n.A[s@n/x]) \to A[\mathsf{lam}\ s/x]\end{array}}{\Gamma \vdash \mathsf{rec}_A(M,N,R) : \Pi x\colon \mathsf{Lam}.\, A}$$

The behaviour of the recursion constants introduced by this rule is specified by the equations below. In the rules for these equations, we abbreviate $\mathsf{rec}_A(M,N,R)$ by rec and write '…' for the premises of the above rule (Lam-REC).

$$\frac{\Gamma \vdash n : \mathbf{N} \qquad \ldots}{\Gamma \vdash \mathsf{rec}\ (\mathsf{var}\ n) = M\ n : A[\mathsf{var}\ n/x]}$$

$$\frac{\Gamma \vdash s : \mathsf{Lam} \qquad \Gamma \vdash t : \mathsf{Lam} \qquad \ldots}{\Gamma \vdash \mathsf{rec}\ (\mathsf{app}\ s\ t) = N\ s\ t\ (\mathsf{rec}\ s)\ (\mathsf{rec}\ t) : A[\mathsf{app}\ s\ t/x]}$$

$$\frac{\Gamma \vdash s : \mathrm{H}n.\,\mathsf{Lam} \qquad \ldots}{\Gamma \vdash \mathsf{rec}\ (\mathsf{lam}\ s) = R\ s\ (\lambda^{\mathrm{H}} n.\,\mathsf{rec}\ (s@_{\mathrm{H}} n)) : A[\mathsf{lam}\ s/x]}$$

The recursion principle and its equations are what one would expect when viewing the H-types in the inductive definition of Lam as the function space Π^*. We justify the recursion principle semantically in Section 12.1.4 below.

We use H-types to represent α-equivalence classes. This gives us two ways of working with them: as partial functions Π^* and as pairs-with-hiding Σ^*. For example, there are two ways to represent the untyped λ-term $(\lambda n.m\ n)$. Using the Π^*-view, it can be represented by

$$m\colon \mathbf{N} \vdash \mathsf{lam}\ (\lambda^{\mathrm{H}} n.\,\mathsf{app}\ (\mathsf{var}\ m)\ (\mathsf{var}\ n)) : \mathsf{Lam}.$$

This representation is in the style of Weak Higher Order Abstract Syntax. Using the Σ^*-view, the same term can also be represented by

$$m\colon \mathbf{N} * n\colon \mathbf{N} \vdash \mathsf{lam}\ (n.(\mathsf{app}\ (\mathsf{var}\ m)\ (\mathsf{var}\ n))) : \mathsf{Lam}.$$

Note that we need an additional fresh name n in order to form this term. The Σ^*-representation is like the representation of syntax in FM-theory. Using H-types, we can thus encode syntax both in WHOAS and in FM-style—at the same time.

Of course, both representations of the term should be equal, since they represent the same object-level term. This is indeed available in the theory, since, along the lines of the argument in 11.3.3.3, we can show the equality

$$m\colon \mathbf{N} * n\colon \mathbf{N} \vdash (n.(\mathsf{app}\ (\mathsf{var}\ m)\ (\mathsf{var}\ n))) = \lambda^{\mathrm{H}} n.\,\mathsf{app}\ (\mathsf{var}\ m)\ (\mathsf{var}\ n) : \mathrm{H}n.\,\mathsf{Lam}.$$

The point that syntax can be represented in the styles of WHOAS and FM, and that these styles can be mixed, is further illustrated in the next two sections in which we give examples for functions that can be defined for Lam.

12.1.1 Substitution

As a first example of a recursively defined function, we define capture-avoiding substitution.

$$\vdash \mathsf{subst} : \mathsf{Lam} \to \mathbf{N} \to \mathsf{Lam} \to \mathsf{Lam}$$

The application $(\mathsf{subst}\ s\ n\ t)$ is intended to denote the result of substituting the term s for the name n in the term t. The recursion steps are given by the following definitions.

$$x\colon \mathsf{Lam}, n\colon \mathbf{N} \vdash \begin{array}{l} M \stackrel{\text{def}}{=} (\lambda m\colon \mathbf{N}.\,\text{ifeq}\ \langle m,n\rangle\ \text{then}\ x\ \text{else}\ (\mathsf{var}\ m)) \\ \qquad\qquad : \Pi m\colon \mathbf{N}.\,\mathsf{Lam} \end{array}$$

$$x\colon \mathsf{Lam}, n\colon \mathbf{N} \vdash \begin{array}{l} N \stackrel{\text{def}}{=} (\lambda s\colon \mathsf{Lam}.\,\lambda t\colon \mathsf{Lam}.\,\lambda h_s\colon \mathsf{Lam}.\,\lambda h_t\colon \mathsf{Lam}.\,\mathsf{app}\ h_s\ h_t) \\ \qquad\qquad : \mathsf{Lam} \to \mathsf{Lam} \to \mathsf{Lam} \to \mathsf{Lam} \to \mathsf{Lam} \end{array}$$

$$x\colon \mathsf{Lam}, n\colon \mathbf{N} \vdash \begin{array}{l} R \stackrel{\text{def}}{=} (\lambda s\colon (\mathrm{H}n.\,\mathsf{Lam}).\,\lambda h_s\colon (\mathrm{H}n.\,\mathsf{Lam}).\,\mathsf{lam}\ h_s) \\ \qquad\qquad : (\mathrm{H}n.\,\mathsf{Lam}) \to (\mathrm{H}n.\,\mathsf{Lam}) \to \mathsf{Lam} \end{array}$$

Given these definitions, we can use the recursion principle to define substitution.

$$\vdash \mathsf{subst} \stackrel{\text{def}}{=} \lambda x\colon \mathsf{Lam}.\,\lambda n\colon \mathbf{N}.\,\mathsf{rec}_{\mathsf{Lam}}(M,N,R) : \mathsf{Lam} \to \mathbf{N} \to \mathsf{Lam} \to \mathsf{Lam}$$

The so defined term satisfies the following equations.

$$\begin{aligned} \mathsf{subst}\ u\ n\ (\mathsf{var}\ m) &= \text{ifeq}\ \langle m,n\rangle\ \text{then}\ u\ \text{else}\ (\mathsf{var}\ m) \\ \mathsf{subst}\ u\ n\ (\mathsf{app}\ s\ t) &= \mathsf{app}\ (\mathsf{subst}\ u\ n\ s)\ (\mathsf{subst}\ u\ n\ t) \\ \mathsf{subst}\ u\ n\ (\mathsf{lam}\ s) &= \mathsf{lam}\ (\lambda^{\mathrm{H}} m.\,\mathsf{subst}\ u\ n\ (s@_{\mathrm{H}} m)) \end{aligned}$$

This definition of substitution uses only the view of H as the function type Π^*. It is close to definitions in Weak Higher Order Abstract Syntax.

In their work on FM set theory, Gabbay & Pitts [38] emphasise the view of binding in terms of the abstraction set, corresponding to the Σ^*-view of H in our terminology. In particular, in [38] substitution is defined using only the operations of the abstraction set. By viewing H as Σ^*, we can also give a definition of substitution in this style. The recursion cases for variables and application are given as before by the terms M and N above. For the λ-binder, we now take the following recursion step.

$$x\colon \mathsf{Lam}, n\colon \mathbf{N} \vdash \begin{array}{ll} R' \stackrel{\text{def}}{=} & \big(\lambda s\colon (\mathrm{H}n.\,\mathsf{Lam}).\,\lambda h_s\colon (\mathrm{H}n.\,\mathsf{Lam}). \\ & \quad \text{new}\ m.\,\text{let bind}(m, h_s@_{\mathrm{H}} m)\ \text{be}\ y^{*m}\ \text{in}\ (\mathsf{lam}\ y)^{*m}\big) \\ & \qquad : (\mathrm{H}n.\,\mathsf{Lam}) \to (\mathrm{H}n.\,\mathsf{Lam}) \to \mathsf{Lam} \end{array}$$

If we define subst by $\lambda x\colon \mathsf{Lam}.\,\lambda n\colon \mathbf{N}.\,\mathsf{rec}_{\mathsf{Lam}}(M,N,R')$ then it satisfies the following equations.

$$\begin{aligned} \mathsf{subst}\ u\ n\ (\mathsf{var}\ m) &= \text{ifeq}\ \langle m,n\rangle\ \text{then}\ u\ \text{else}\ (\mathsf{var}\ m) \\ \mathsf{subst}\ u\ n\ (\mathsf{app}\ s\ t) &= \mathsf{app}\ (\mathsf{subst}\ u\ n\ s)\ (\mathsf{subst}\ u\ n\ t) \\ \mathsf{subst}\ u\ n\ (\mathsf{lam}\ s) &= \text{new}\ m.\,\text{let bind}(m, \mathsf{subst}\ u\ n\ (s@_{\mathrm{H}} m))\ \text{be}\ y^{*m}\ \text{in}\ (\mathsf{lam}\ y)^{*m} \end{aligned}$$

By the equations for new from Section 11.3.3.1, we further have:

$$\mathsf{subst}\ u\ n\ (\mathsf{lam}\ s) = \text{let}\ s\ \text{be}\ m.x\ \text{in}\ (\text{let bind}(m, \mathsf{subst}\ u\ n\ x)\ \text{be}\ y^{*m}\ \text{in}\ (\mathsf{lam}\ y)^{*m})$$

This uses only the Σ^*-view of H and corresponds to a definition in the style of FM.

Having given two definitions of substitution, it is natural to ask if they define the same function. It is not hard see by unwinding the semantic interpretation that both define the intended substitution function. Furthermore, the two definitions are provably equal, which can be seen by the following chain of equations.

$$\begin{aligned}
&\text{new } m.\,\text{let bind}(m, \mathsf{subst}\ u\ n\ (s@_{\mathrm{H}}m))\ \text{be}\ y^{*m}\ \text{in}\ (\mathsf{lam}\ y)^{*m} \\
&= \text{let}\ (\lambda^{\mathrm{H}}m.\,\text{unit})\ \text{be}\ m.v\ \text{in let bind}(m, \mathsf{subst}\ u\ n\ (s@_{\mathrm{H}}m))\ \text{be}\ y^{*m}\ \text{in}\ (\mathsf{lam}\ y)^{*m} && \text{by defn.} \\
&= \text{let}\ (\lambda^{\mathrm{H}}m.\,\mathsf{subst}\ u\ n\ (s@_{\mathrm{H}}m))\ \text{be}\ m.x\ \text{in let bind}(m,x)\ \text{be}\ y^{*m}\ \text{in}\ (\mathsf{lam}\ y)^{*m} && \text{by (H-NAT)} \\
&= \mathsf{lam}\ (\lambda^{\mathrm{H}}m.\,\mathsf{subst}\ u\ n\ (s@_{\mathrm{H}}m)) && \text{by (H-}\eta\text{-}\Sigma^*\text{)}
\end{aligned}$$

Generalising the FM-style definition of substitution, we show that the following FM-style recursion principle is derivable.

$$\frac{\begin{array}{l}\vdash A\ \text{Type} \\ \Gamma \vdash M : \mathbf{N} \to A \\ \Gamma \vdash N : \mathsf{Lam} \to \mathsf{Lam} \to A \to A \to A \\ \Gamma \vdash R : \mathrm{H}n.\,\mathsf{Lam} \to A \to \left(A^{*(n:\mathbf{N})}\right)\end{array}}{\Gamma \vdash \mathsf{rec}'_A(M,N,R) : \Pi x\colon \mathsf{Lam}.\,A}$$

The restriction to closed types A is necessary for the type $(A^{*(n:\mathbf{N})})$ to be well-formed.

When one compares this recursion principle to the corresponding one in FM set theory [38, Corollary 6.7], one finds that instead of a term R of type $\mathrm{H}n.\,\mathsf{Lam} \to A \to (A^{*(n:\mathbf{N})})$, the FM recursion principle requires one to give a function $f\colon \mathbf{N} \to \mathsf{Lam} \to A \to A$ that satisfies the freshness property $И n.\,\forall t\colon \mathsf{Lam}.\,\forall x\colon A.\,f(n,t,x) \# n$. In the above recursion principle, the type of R includes the freshness property that f must satisfy. Semantically, if we have in FM set theory a function f satisfying the freshness property, then we can construct a term R of type $\mathrm{H}n.\,\mathsf{Lam} \to A \to \left(A^{*(n:\mathbf{N})}\right)$ from it.

The rec'-equations for the var-case and the app-case are the same as for rec. The equation for the lam-case, in which we abbreviate $\mathsf{rec}'_A(M,N,R)$ by rec', appears below.

$$\frac{\Gamma \vdash s : \mathrm{H}n.\,\mathsf{Lam} \qquad \ldots}{\Gamma \vdash \mathsf{rec}'\ (\mathsf{lam}\ s) = \text{let}\ s\ \text{be}\ n.x\ \text{in}\ (R@_{\mathrm{H}}n)\ x\ (\mathsf{rec}'\ x) : A}$$

This equation formalises that in the recursion case for a binder we use a form of pattern-matching. The α-equivalence class s is matched against a pair $n.x$ of a fresh name n and the instance x of the α-equivalence class s at the name n.

The recursion principle rec' can be defined from rec as follows. Suppose we have terms

$$\begin{array}{l}\Gamma \vdash M : \mathbf{N} \to A \\ \Gamma \vdash N : \mathsf{Lam} \to \mathsf{Lam} \to A \to A \to A \\ \Gamma \vdash R : \mathrm{H}n.\,\mathsf{Lam} \to A \to \left(A^{*(n:\mathbf{N})}\right).\end{array}$$

We need to define a term

$$\Gamma \vdash \mathsf{rec}'_A(M,N,R) : \Pi x\colon \mathsf{Lam}.A.$$

To this end, define the following term.

$$\Gamma \vdash R' \stackrel{\text{def}}{=} \lambda s\colon (\mathrm{H}n.\mathsf{Lam}).\lambda t\colon (\mathrm{H}n.A).\,\mathsf{new}\ n.(R@_{\mathrm{H}}n)\ (s@_{\mathrm{H}}n)\ (t@_{\mathrm{H}}n) : (\mathrm{H}n.\mathsf{Lam}) \to (\mathrm{H}n.A) \to A$$

Note that the freefrom type in the codomain of R is essential for R' to be typeable; see the rules for new in Section 11.3.3.1. Define $\mathsf{rec}'_A(M,N,R)$ to be $\mathsf{rec}_A(M,N,R')$. From the equations for rec, it follows immediately that the required equations for the var and app-cases hold with this definition. For the lam-case, the corresponding equation for rec gives:

$$\mathsf{rec}'\ (\mathsf{lam}\ s) = \mathsf{new}\ n.(R@_{\mathrm{H}}n)\ (s@_{\mathrm{H}}n)\ (\mathsf{rec}'\ (s@_{\mathrm{H}}n))$$

We have the following series of equations:

$$\begin{aligned}
\mathsf{rec}'\ (\mathsf{lam}\ s) &= \mathsf{new}\ n.(R@_{\mathrm{H}}n)\ (s@_{\mathrm{H}}n)\ (\mathsf{rec}'\ (s@_{\mathrm{H}}n)) && \text{by defn.}\\
&= \mathsf{let}\ (\lambda^{\mathrm{H}}n.\mathrm{unit})\ \mathsf{be}\ n.u\ \mathsf{in}\ (R@_{\mathrm{H}}n)\ (s@_{\mathrm{H}}n)\ (\mathsf{rec}'\ (s@_{\mathrm{H}}n)) && \text{by defn.}\\
&= \mathsf{let}\ (\lambda^{\mathrm{H}}n.s@_{\mathrm{H}}n)\ \mathsf{be}\ n.x\ \mathsf{in}\ (R@_{\mathrm{H}}n)\ x\ (\mathsf{rec}'\ x) && \text{by (H-NAT)}\\
&= \mathsf{let}\ s\ \mathsf{be}\ n.x\ \mathsf{in}\ (R@_{\mathrm{H}}n)\ x\ (\mathsf{rec}'\ x) && \text{by } (\Pi^*\text{-}\eta)
\end{aligned}$$

Hence, the term rec′ satisfies the required equations.

12.1.2 Free Variables

As a second example of a recursively defined function, we define a function that computes the free variables of a term. To represent the set of free names, we use an inductively defined type ($\vdash \mathsf{LN}\ \mathsf{Type}$) of lists of names, as defined in Section 11.1.1. We assume appropriately defined terms

$$\vdash \mathsf{singleton} : \mathbf{N} \to \mathsf{LN}$$

$$\vdash \mathsf{union} : \mathsf{LN} \to \mathsf{LN} \to \mathsf{LN}$$

$$\vdash \mathsf{remove} : \Pi n\colon \mathbf{N}.\,\mathsf{LN} \to \left(\mathsf{LN}^{*(n:\mathbf{N})}\right)$$

with the expected meaning. By defining

$$\vdash M \stackrel{\text{def}}{=} (\lambda m\colon \mathbf{N}.\,\mathsf{singleton}\ m)\ :\ \Pi m\colon \mathbf{N}.\,\mathsf{LN}$$

$$\vdash N \stackrel{\text{def}}{=} (\lambda s\colon \mathsf{Lam}.\lambda t\colon \mathsf{Lam}.\lambda h_s\colon \mathsf{LN}.\lambda h_t\colon \mathsf{LN}.\,\mathsf{union}\ h_s\ h_t)\ :\ \mathsf{Lam} \to \mathsf{Lam} \to \mathsf{LN} \to \mathsf{LN} \to \mathsf{LN}$$

$$\vdash R \stackrel{\text{def}}{=} (\lambda s\colon (\mathrm{H}n.\mathsf{Lam}).\lambda h_s\colon (\mathrm{H}n.\mathsf{LN}).\,\mathsf{new}\ n.\mathsf{remove}\ n\ (h_s@n))\ :\ (\mathrm{H}n.\mathsf{Lam}) \to (\mathrm{H}n.\mathsf{LN}) \to \mathsf{LN},$$

and by letting $\vdash \mathsf{fv} \stackrel{\text{def}}{=} \mathsf{rec}_{\mathsf{LN}}(M,N,R) : \mathsf{Lam} \to \mathsf{LN}$, we obtain a function fv that satisfies

$$\begin{aligned}
\mathsf{fv}\ (\mathsf{var}\ n) &= \mathsf{singleton}\ n\\
\mathsf{fv}\ (\mathsf{app}\ s\ t) &= \mathsf{union}\ (\mathsf{fv}\ s)\ (\mathsf{fv}\ t)\\
\mathsf{fv}\ (\mathsf{lam}\ s) &= \mathsf{new}\ n.\mathsf{remove}\ n\ (\mathsf{fv}\ (s@n)).
\end{aligned}$$

The recursion case for the lam-binder is another example where a freefrom type is essential for typeability. The term $(\text{new } n.\,\mathsf{remove}\ n\ (\mathsf{fv}\ (s@n)))$ is typeable only because of the freefrom type in the type of remove. The intuition is that, when given the argument $(\mathsf{lam}\ s)$, the function fv picks a fresh name, computes $(s@n)$ and then removes the fresh name from the result. Since our language does not have side effects, the definition of fv must contain the information that the choice of the name n does not affect the outcome of the computation. This information is contained in the freefrom type.

Again, there are other possibilities for defining the function fv. For example, we can define fv so that it satisfies the following equations.

$$\begin{aligned}\mathsf{fv}\ (\mathsf{var}\ n) &= \mathsf{singleton}\ n\\ \mathsf{fv}\ (\mathsf{app}\ s\ t) &= \mathsf{union}\ (\mathsf{fv}\ s)\ (\mathsf{fv}\ t)\\ \mathsf{fv}\ (\mathsf{lam}\ s) &= \mathsf{let}\ (\lambda^{\mathrm{H}} n.\,(\mathsf{fv}\ (s@n))\ \text{be}\ n.x\ \text{in}\ (\mathsf{remove}\ n\ x)\end{aligned}$$

The last equation is another example where it is useful to mix the two views of H as Σ^* and Π^*. In effect, in this equation we take a function and pattern-match it against a pair. That the two ways of defining fv result in provably equal functions can be shown in the same way as for substitution.

To give an example of an evaluation of fv, we show the following simple equation.

$$n\colon \mathbf{N} \vdash (\mathsf{fv}\ (\mathsf{lam}\ (\lambda^{\mathrm{H}} m.\,\mathsf{app}\ (\mathsf{var}\ n)\ (\mathsf{var}\ m)))) = \mathsf{cons}\ n\ \mathsf{nil} : \mathsf{LN}$$

To see that this equation holds, first note that we have

$$n\colon \mathbf{N} * m\colon \mathbf{N} \vdash (\mathsf{fv}\ (\mathsf{app}\ (\mathsf{var}\ n)\ (\mathsf{var}\ m))) = \mathsf{cons}\ n\ (\mathsf{cons}\ m\ \mathsf{nil}) : \mathsf{LN},$$

as is not hard to show. In Section 11.1.1, we have shown the equation

$$n\colon \mathbf{N} * m\colon \mathbf{N} \vdash \mathsf{remove}\ m\ (\mathsf{cons}\ n\ (\mathsf{cons}\ m\ \mathsf{nil})) = (\mathsf{cons}\ n\ \mathsf{nil})^{*m} : \mathsf{LN}^{*(m:\mathbf{N})}.$$

The required equation can therefore be calculated as follows.

$$\begin{aligned}&\mathsf{fv}\ (\mathsf{lam}\ (\lambda^{\mathrm{H}} m.\,\mathsf{app}\ (\mathsf{var}\ n)\ (\mathsf{var}\ m)))\\ &= \text{new } m.\,\mathsf{remove}\ m\ (\mathsf{fv}\ (\mathsf{app}\ (\mathsf{var}\ n)\ (\mathsf{var}\ m))) && \text{by defn.}\\ &= \text{new } m.\,\mathsf{remove}\ m\ (\mathsf{cons}\ n\ (\mathsf{cons}\ m\ \mathsf{nil})) && \text{using above eqn.}\\ &= \text{new } m.\,(\mathsf{cons}\ n\ \mathsf{nil})^{*m} && \text{using above eqn.}\\ &= \mathsf{cons}\ n\ \mathsf{nil} && \text{see Section 11.3.3.1}\end{aligned}$$

12.1.3 Recursion over Term Contexts

The Theory of Contexts [51] features recursion not only for terms but also for term contexts, i.e. terms with holes. In our setting, this corresponds to induction principles over $\mathrm{H}n.\,\mathsf{Lam}$ and $\mathrm{H}n.\,\mathrm{H}m.\,\mathsf{Lam}$ etc.

We consider the recursion principle over $\mathrm{H}n.\,\mathsf{Lam}$, looking at a simplified form with closed types.

$$\frac{\vdash A\ \mathrm{Type} \qquad \begin{array}{lll} \Gamma \vdash M : A & & \Gamma \vdash R : A \to A \to A \\ \Gamma \vdash N : \mathbf{N} \to A & & \Gamma \vdash S : (\mathrm{H}n.A) \to A \end{array}}{\Gamma \vdash \mathsf{rec1}_A(M,N,R,S) : (\mathrm{H}n.\mathsf{Lam}) \to A}$$

The equations for this principle, in which we write $\mathsf{rec1}$ for $\mathsf{rec1}_A(M,N,R,S)$ and in which we omit the typing assumptions, are as follows.

$$\begin{aligned} \mathsf{rec1}\ (\lambda^{\mathrm{H}} n.\mathsf{var}\ n) &= M \\ \mathsf{rec1}\ (\lambda^{\mathrm{H}} n.\mathsf{var}\ m) &= N\ m && n \neq m \\ \mathsf{rec1}\ (\lambda^{\mathrm{H}} n.\mathsf{app}\ (s@_{\mathrm{H}} n)\ (t@_{\mathrm{H}} n)) &= R\ (\mathsf{rec1}\ s)\ (\mathsf{rec1}\ t) \\ \mathsf{rec1}\ (\lambda^{\mathrm{H}} n.\mathsf{lam}\ (s@_{\mathrm{H}} n)) &= S\ (\lambda^{\mathrm{H}} m.\mathsf{rec1}\ (\lambda^{\mathrm{H}} n.s@_{\mathrm{H}} n@_{\mathrm{H}} m)) \end{aligned}$$

For the derivation of such a recursion principle, we need a term

$$n\colon \mathbf{N}, x\colon \mathrm{H}m.\left(\mathsf{Lam}^{*(n:\mathbf{N})}\right) \vdash O : (\mathrm{H}m.\mathsf{Lam})^{*(n:\mathbf{N})}$$

satisfying the equation

$$n\colon \mathbf{N}, x\colon \mathrm{H}m.\left(\mathsf{Lam}^{*(n:\mathbf{N})}\right) \vdash (\text{let } O \text{ be } y^{*n} \text{ in } y) = (\lambda^{\mathrm{H}} m.\text{let } (x@_{\mathrm{H}} m) \text{ be } y^{*n} \text{ in } y) : \mathrm{H}m.\mathsf{Lam}.$$

In Section 11.3.3.4, we have shown how open freefrom types can be used to derive such a term. We now assume that we have such a term.

We start the derivation of $\mathsf{rec1}$ from the bottom, i.e. we define $\mathsf{rec1}$ as the term in the conclusion of the following derivation.

$$\frac{\dfrac{(\Gamma * n\colon \mathbf{N}), x\colon \mathsf{Lam} \vdash Q : A^{*(n:\mathbf{N})}}{\Gamma, z\colon \mathrm{H}n.\mathsf{Lam} \vdash \text{let } z \text{ be } n.x \text{ in } Q : A}}{\Gamma \vdash \lambda z\colon \mathrm{H}n.\mathsf{Lam}.\text{let } z \text{ be } n.x \text{ in } Q : (\mathrm{H}n.\mathsf{Lam}) \to A}$$

We note that the assumption that A be a closed type is needed for being able to form the type $A^{*(n:\mathbf{N})}$. We define Q by normal recursion over $x\colon \mathsf{Lam}$.

In the base case, in which x represents a variable $(\mathsf{var}\ m)$, the term Q should be given essentially by $(\text{if } m = n \text{ then } M \text{ else } N)$. But note that the type of Q is $A^{*(n:\mathbf{N})}$, so that we also have to establish a freshness property. We define a suitable term using ifeq. In order to deal with the dependencies of the freefrom types, we first define a term of the following form.

$$(\Gamma * n\colon \mathbf{N}), m\colon \mathbf{N} \vdash \text{ifeq } \langle n,m\rangle \text{ then } r.Q_e \text{ else } q.Q_d : A^{*(n:\mathbf{N})} \to (\mathbf{N} \to A)^{*(n:\mathbf{N})} \to A^{*(n:\mathbf{N})}$$

This term is then applied to $M^{*n}\colon A^{*(n:\mathbf{N})}$ and $N^{*n}\colon (\mathbf{N} \to A)^{*(n:\mathbf{N})}$ to obtain the required term. We write Γ' for $(\Gamma * n\colon \mathbf{N}), m\colon \mathbf{N}$ and $A'(n)$ for $A^{*(n:\mathbf{N})} \to (\mathbf{N} \to A)^{*(n:\mathbf{N})} \to A^{*(n:\mathbf{N})}$.

The branch of the equality test in which the names are equal is given by the following term.

$$n\colon \mathbf{N} \vdash Q_e \stackrel{\text{def}}{=} \lambda y\colon A^{*(n:\mathbf{N})}.\lambda x\colon (\mathbf{N} \to A)^{*(n:\mathbf{N})}.y\ :\ A^{*(n:\mathbf{N})} \to (\mathbf{N} \to A)^{*(n:\mathbf{N})} \to A^{*(n:\mathbf{N})}$$

The term for the branch where the names differ is given by the following derivation.

$$\dfrac{\dfrac{\dfrac{(n\colon \mathbf{N} * m\colon \mathbf{N}), x\colon (\mathbf{N}\to A)^{*(n:\mathbf{N})} \vdash \mathrm{join}(x, m^{*n}) : ((\mathbf{N}\to A)\times\mathbf{N})^{*(n:\mathbf{N})} \quad (y\colon (\mathbf{N}\to A), m\colon \mathbf{N}) * n\colon \mathbf{N} \vdash (y\ m)^{*n} : A^{*(n:\mathbf{N})}}{(n\colon \mathbf{N} * m\colon \mathbf{N}), x\colon (\mathbf{N}\to A)^{*(n:\mathbf{N})} \vdash \text{let } \mathrm{join}(x, m^{*n}) \text{ be } \langle y, m\rangle^{*n} \text{ in } (y\ m)^{*n} : A^{*(n:\mathbf{N})}}}{(n\colon \mathbf{N} * m\colon \mathbf{N}) \vdash \lambda x\colon (\mathbf{N}\to A)^{*(n:\mathbf{N})}.\,\text{let } \mathrm{join}(x, m^{*n}) \text{ be } \langle y, m\rangle^{*n} \text{ in } (y\ m)^{*n} : A^{*(n:\mathbf{N})}}}{(n\colon \mathbf{N} * m\colon \mathbf{N}), y\colon A^{*(n:\mathbf{N})} \vdash \lambda x\colon (\mathbf{N}\to A)^{*(n:\mathbf{N})}.\,\text{let } \mathrm{join}(x, m^{*n}) \text{ be } \langle y, m\rangle^{*n} \text{ in } (y\ m)^{*n} : (\mathbf{N}\to A)^{*(n:\mathbf{N})} \to A^{*(n:\mathbf{N})}}}{n\colon \mathbf{N} * m\colon \mathbf{N} \vdash \lambda y\colon A^{*(n:\mathbf{N})}.\,\lambda x\colon (\mathbf{N}\to A)^{*(n:\mathbf{N})}.\,\text{let } \mathrm{join}(x, m^{*n}) \text{ be } \langle y, m\rangle^{*n} \text{ in } (y\ m)^{*n} : A'(n)}$$

Let Q_d be the term in the conclusion of this derivation. With these definitions of Q_e and Q_d, we have:

$$\dfrac{\dfrac{n\colon \mathbf{N} \vdash Q_e : A'(n)}{\Gamma', n\colon \mathbf{N} \vdash Q_e : A'(n)} \qquad \dfrac{\dfrac{n\colon \mathbf{N} * m\colon \mathbf{N} \vdash Q_d : A'(n)}{q\colon \mathbf{N} * \mathbf{N} \vdash \text{let } q \text{ be } n * m \text{ in } Q_d : A'(\hat{\pi}_1(q))}}{\Gamma', q\colon \mathbf{N} * \mathbf{N} \vdash \text{let } q \text{ be } n * m \text{ in } Q_d : A'(\hat{\pi}_1(q))}}{\Gamma' \vdash \text{ifeq } \langle n, m\rangle \text{ then } r.\,Q_e \text{ else } q.\,Q_d : A'(n)}$$

Since we can make the judgements

$$(\Gamma * n\colon \mathbf{N}), m\colon \mathbf{N} \vdash M^{*n} : A^{*(n:\mathbf{N})}$$
$$(\Gamma * n\colon \mathbf{N}), m\colon \mathbf{N} \vdash N^{*n} : (\mathbf{N}\to A)^{*(n:\mathbf{N})},$$

we can therefore derive

$$\Gamma * n\colon \mathbf{N} \vdash \lambda m\colon \mathbf{N}.\,(\text{ifeq } \langle n, m\rangle \text{ then } r.\,Q_e \text{ else } q.\,Q_d)\ M^{*n}\ N^{*n}\ :\ \Pi m\colon \mathbf{N}.\,A^{*(n:\mathbf{N})}.$$

We write M' for this term. It is the case for a variable in the recursive definition.

For the recursion case of an application $(\mathsf{app}\ s\ t)$, we simply use the induction hypothesis. Using a suggestive notation for (defined) ternary join, we have

$$(\Gamma * n\colon \mathbf{N}), h_s\colon A^{*(n:\mathbf{N})}, h_t\colon A^{*(n:\mathbf{N})} \vdash \text{let } \mathrm{join}(R^{*n}, h_s, h_t) \text{ be } \langle u, v, w\rangle^{*n} \text{ in } (u\ v\ w)^{*n}\ :\ A^{*(n:\mathbf{N})}.$$

Writing Q_a for this term, we have

$$\Gamma * n\colon \mathbf{N} \vdash \lambda s, t\colon \mathsf{Lam}.\,\lambda h_s, h_t\colon A^{*(n:\mathbf{N})}.\,Q_a\ :\ \mathsf{Lam}\to\mathsf{Lam}\to A^{*(n:\mathbf{N})}\to A^{*(n:\mathbf{N})}\to A^{*(n:\mathbf{N})}.$$

This term is the application case of the recursive definition. We write N' for it.

It remains to do the case for a lambda abstraction $(\mathsf{lam}\ s)$. For this case we need the term O that we have assumed. The reason for this is that the recursion hypothesis gives us an element of type $\mathrm{H}m.\,\left(A^{*(n:\mathbf{N})}\right)$, and using the term $S\colon (\mathrm{H}n.A)\to A$ we would like to obtain an element of type $A^{*(n:\mathbf{N})}$ from it. We can do this using the term O, since it allows us to go from $\mathrm{H}m.\,\left(A^{*(n:\mathbf{N})}\right)$ to $(\mathrm{H}m.A)^{*(n:\mathbf{N})}$.

$$(\Gamma * n\colon \mathbf{N}), x\colon \mathrm{H}m.\,\left(A^{*(n:\mathbf{N})}\right) \vdash \text{let } \mathrm{join}(S^{*n}, O) \text{ be } \langle u, v\rangle^{*n} \text{ in } (u\ v)^{*n} : A^{*(n:\mathbf{N})}$$

Writing Q_l for the term in this sequent, we have

$$\Gamma * n\colon \mathbf{N} \vdash \lambda s\colon (\mathrm{H}n.\mathsf{Lam}).\,\lambda x\colon \left(\mathrm{H}m.\,\left(A^{*(n:\mathbf{N})}\right)\right).\,Q_l\ :\ (\mathrm{H}n.\mathsf{Lam})\to\left(\mathrm{H}m.\,\left(A^{*(n:\mathbf{N})}\right)\right)\to A^{*(n:\mathbf{N})}.$$

This term is the case for the lambda-binder in the recursive definition. We write R' for it.

Having defined the three cases for the recursive definition, we can define Q to be the following term.

$$(\Gamma * n\colon \mathbf{N}), x\colon \mathsf{Lam} \vdash \mathsf{rec}_{(A^{*(n:\mathbf{N})})}(M', N', R'))\ x : A^{*(n:\mathbf{N})}$$

This completes the definition of the recursion principle.

12.1.4 Soundness of the Recursion Principle

In this section we justify the recursion principle for Lam semantically. We start with the special case of the recursion principle in which Γ is the empty context.

Consider the endo-functor F on the Schanuel topos defined by

$$F(X) \stackrel{\text{def}}{=} \mathbf{N} + (X \times X) + (\mathbf{N} \multimap X).$$

It is well-known, see e.g. [49, 38, 30], that F has an initial algebra $[\mathsf{var}, \mathsf{app}, \mathsf{lam}]\colon F(\mathsf{Lam}) \to \mathsf{Lam}$. As suggested by the notation, the type Lam and its constructors are interpreted by this initial algebra.

It remains to interpret the recursion principle (Lam-REC). In the premises of (Lam-REC), we have a type $x\colon \mathsf{Lam} \vdash A$ Type, and we write $A(s)$ as a short-hand for $A[s/x]$. Furthermore, we have (up to unique correspondence) terms of the following types.

$$\begin{aligned} n\colon \mathbf{N} &\vdash M : A(\mathsf{var}\ n) \\ s\colon \mathsf{Lam}, t\colon \mathsf{Lam} &\vdash N : A(s) \to A(t) \to A(\mathsf{app}\ s\ t) \\ s\colon (\mathrm{H}n.\mathsf{Lam}), h\colon (\mathrm{H}n.A(s@n)) &\vdash R : A(\mathsf{lam}\ s) \end{aligned}$$

It is straightforward to see that these terms correspond uniquely to a morphism u making the following diagram in $\mathbb{S}$ commute. Note that we are now viewing the H-type as a $\Pi^*_{\mathbf{N}}$-type.

$$\begin{array}{ccc} \mathbf{N} + (\Sigma p\colon \mathsf{Lam} \times \mathsf{Lam}.A(\mathrm{fst}(p)) \times A(\mathrm{snd}(p))) & & \\ \quad + (\Sigma s\colon \mathbf{N} \multimap \mathsf{Lam}.\Pi^* n\colon \mathbf{N}.A(s@n)) & \xrightarrow{u} & \Sigma s\colon \mathsf{Lam}.A(s) \\ \downarrow{\scriptstyle \mathbf{N}+\pi_1+\pi_1} & & \downarrow{\scriptstyle \pi_1} \\ F(\mathsf{Lam}) & \xrightarrow[[\mathsf{var},\mathsf{app},\mathsf{lam}]]{} & \mathsf{Lam} \end{array}$$

We now show that the object in the upper left corner of this diagram is isomorphic to $F(\Sigma s\colon \mathsf{Lam}.A(s))$. To this end, we first have an evident isomorphism

$$\Sigma p\colon \mathsf{Lam} \times \mathsf{Lam}.A(\mathrm{fst}(p)) \times A(\mathrm{snd}(p)) \ \cong\ (\Sigma s\colon \mathsf{Lam}.A(s)) \times (\Sigma t\colon \mathsf{Lam}.A(t)).$$

Furthermore, there is an isomorphism

$$\Sigma s\colon \mathbf{N} \multimap \mathsf{Lam}.\Pi^* n\colon \mathbf{N}.A(s@n) \ \cong\ \mathbf{N} \multimap (\Sigma t\colon \mathsf{Lam}.A(t))$$

corresponding to a version of the type-theoretic 'axiom of choice'. From left to right, this isomorphism is given by the term

$$s\colon \mathbf{N} \multimap \mathsf{Lam},\ f\colon \Pi^* n\colon \mathbf{N}.A(s@n) \vdash \lambda^* n\colon \mathbf{N}.\langle s@n, f@n\rangle \ :\ \mathbf{N} \multimap (\Sigma s\colon \mathsf{Lam}.A(s)),$$

and from right to left by the term

$$f\colon \mathbf{N} \multimap (\Sigma t\colon \mathsf{Lam}.A(t)) \vdash \langle \lambda^* n\colon \mathbf{N}.\mathrm{fst}(f@n), \lambda^* n\colon \mathbf{N}.\mathrm{snd}(f@n)\rangle \ :\ \Sigma s\colon \mathbf{N} \multimap \mathsf{Lam}.\Pi^* n\colon \mathbf{N}.A(s@n).$$

From these two isomorphisms, it follows immediately that the top left corner in the above diagram is isomorphic to $F(\Sigma s\colon \mathsf{Lam}.A(s))$. The morphism u making the above diagram commute therefore corresponds uniquely to a morphism u' making the following diagram commute.

$$\begin{array}{ccc} F(\Sigma s\colon \mathsf{Lam}.A(s)) & \xrightarrow{u'} & \Sigma s\colon \mathsf{Lam}.A(s) \\ {\scriptstyle F(\pi_1)}\downarrow & & \downarrow{\scriptstyle \pi_1} \\ F(\mathsf{Lam}) & \xrightarrow[[\mathsf{var},\mathsf{app},\mathsf{lam}]]{} & \mathsf{Lam} \end{array}$$

The initial algebra $[\mathsf{var}, \mathsf{app}, \mathsf{lam}]$ now gives us a morphism R making the diagram below commute.

$$\begin{array}{ccc} F(\mathsf{Lam}) & \xrightarrow{[\mathsf{var},\mathsf{app},\mathsf{lam}]} & \mathsf{Lam} \\ {\scriptstyle F(R)}\downarrow & & \downarrow{\scriptstyle R} \\ F(\Sigma s\colon \mathsf{Lam}.A(s)) & \xrightarrow{u'} & \Sigma s\colon \mathsf{Lam}.A(s) \\ {\scriptstyle F(\pi_1)}\downarrow & & \downarrow{\scriptstyle \pi_1} \\ F(\mathsf{Lam}) & \xrightarrow[[\mathsf{var},\mathsf{app},\mathsf{lam}]]{} & \mathsf{Lam} \end{array}$$

The uniqueness part of initiality gives us $\pi_1 \circ R = id$, so that R corresponds to a term $s\colon \mathsf{Lam} \vdash \mathsf{rec}\colon A(s)$. That rec satisfies the recursion equations follows because the upper square in the diagram commutes.

The general case of the recursion principle, in which Γ is not necessarily empty, can be reduced to the above special case. Given $\Gamma, x\colon \mathsf{Lam} \vdash A$ Type and $\Gamma \vdash M : \Pi n\colon \mathbf{N}.A(\mathsf{var}\ n)$ etc., we apply the above special case to $x\colon \mathsf{Lam} \vdash \Pi\gamma\colon \lfloor\Gamma\rfloor.A$ Type and $\vdash \lambda n\colon \mathbf{N}.\lambda\gamma\colon \lfloor\Gamma\rfloor.M\ n \ :\ \Pi n\colon \mathbf{N}.\Pi\gamma\colon \lfloor\Gamma\rfloor.A(\mathsf{var}\ n)$ etc.

12.2 Simply-typed λ-calculus

Having defined the syntax of the untyped λ-calculus, we now consider a typing judgement for it. As an example, we consider Curry-style typing [8], for which we can use the syntax of the untyped λ-calculus as it is. We then represent the evaluation of well-typed terms.

12.2.1 Typing Judgement

Object level types are encoded using an inductive type with two constructors:

$$\vdash \mathsf{Ty}\ \mathrm{Type} \qquad \vdash \iota : \mathsf{Ty} \qquad \vdash \Rightarrow : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty}$$

For convenience we use infix notation, writing $(\iota \Rightarrow \iota)$ for $((\Rightarrow\ \iota)\ \iota)$. We omit the standard induction principle and its equations.

Object-level contexts are represented by an inductive type with the following constructors.

$$\vdash \mathsf{Ctx}\ \mathsf{Type} \qquad \vdash \mathsf{empty} : \mathsf{Ctx} \qquad \vdash \mathsf{snoc} : \mathsf{Ctx} \to \mathbf{N} \multimap \mathsf{Ty} \to \mathsf{Ctx}$$

We write $c[n\colon a]$ as an abbreviation for $(((\mathsf{snoc}\ c)@n)\ a)$. The recursion principle is:

$$(\textsf{Ctx-REC})\ \frac{\begin{array}{c}\Gamma, c\colon \mathsf{Ctx} \vdash A\ \mathsf{Type} \\ \Gamma \vdash M : A[\mathsf{empty}/c] \\ \Gamma \vdash N : \Pi c\colon \mathsf{Ctx}.\Pi h\colon A.\Pi^* n\colon \mathbf{N}.\Pi b\colon \mathsf{Ty}.A(c[n\colon b])\end{array}}{\Gamma \vdash \mathsf{rec}_A(M,N) : \Pi c\colon \mathsf{Ctx}.A(c)}$$

Notice the use of $\multimap$ in the constructor snoc. It formalises the common convention that each variable may be declared at most once in each context, i.e. that we can write $c[n\colon a]$ only if n is fresh for c. It appears as though such an inductive definition of Ctx is not directly available in Fresh O'Caml.

To show that the use of $\multimap$ is valid, we justify the inductive definition of Ctx semantically in the Schanuel topos. The constructor snoc corresponds uniquely to a map of type $\mathsf{snoc}\colon (\mathsf{Ctx} * \mathbf{N}) \times \mathsf{Ty} \to \mathsf{Ctx}$. Consequently, we define the inductive type Ctx as the initial algebra of the functor

$$FX \stackrel{\text{def}}{=} 1 + ((X * \mathbf{N}) \times \mathsf{Ty}).$$

This functor has an initial algebra because, being built from $1 + (-)$ and the left adjoints $(-) \times \mathsf{Ty}$ and $(-) * \mathbf{N}$, it preserves colimits, so that its initial coalgebra can be constructed in the usual way as the colimit of the finite iterations of F on 0. To justify the recursion principle, we observe that, for any map u making the lower square in the diagram below commute, there exists a unique map R making the whole diagram commute.

$$\begin{array}{ccc} F(\mathsf{Ctx}) & \xrightarrow{[\mathsf{empty},\mathsf{snoc}]} & \mathsf{Ctx} \\ {\scriptstyle F(R)}\downarrow & & \downarrow{\scriptstyle R} \\ F(\Sigma s\colon \mathsf{Ctx}.A(s)) & \xrightarrow{u} & \Sigma s\colon \mathsf{Ctx}.A(s) \\ {\scriptstyle F(\pi_A)}\downarrow & & \downarrow{\scriptstyle \pi_A} \\ F(\mathsf{Ctx}) & \xrightarrow[[\mathsf{empty},\mathsf{snoc}]]{} & \mathsf{Ctx} \end{array}$$

Hence, to justify the special case of the recursion principle (Ctx-REC) where Γ is the empty context, it suffices to show that the terms M and N correspond uniquely to a map u as in the diagram. By definition of F, it can be seen that u corresponds to two maps in $\mathbb{S}^{\to}$, namely $M_0\colon 1 \to \pi_A$ over $\mathsf{empty}\colon 1 \to \mathsf{Ctx}$ and $N_0\colon \pi_{\mathsf{Ty}}^* W_{\mathbf{N}} \pi_A \to \pi_A$ over $\mathsf{snoc}\colon (\mathsf{Ctx} * \mathbf{N}) \times \mathsf{Ty} \to \mathsf{Ctx}$. These two morphisms correspond to maps $M\colon 1 \to \mathsf{empty}^* \pi_A$ in $\mathbb{S}/1$ and $N_1\colon \pi_{\mathsf{Ty}}^* W_{\mathbf{N}} \pi_A \to \mathsf{snoc}^* \pi_A$ in $\mathbb{S}/(\mathsf{Ctx} * \mathbf{N}) \times \mathsf{Ty}$. By adjoint transpose, the latter corresponds to $N\colon \pi_A \to \Pi_{\mathbf{N}}^* \Pi_{\mathsf{Ty}} \mathsf{snoc}^* \pi_A$ in $\mathbb{S}/\mathsf{Ctx}$. A standard argument then shows that these two maps M and N amount to the terms of the same name in (Ctx-REC). The general case of (Ctx-REC) where Γ is not the empty context can be reduced to the present case, as in Section 12.1.4.

Using the recursion principle we can define an append function $\circ$ on contexts.

$$\vdash \circ : \mathsf{Ctx} \to \mathsf{Ctx} \multimap \mathsf{Ctx}$$

$$c\colon \mathsf{Ctx} \vdash c \circ \mathsf{empty} = c : \mathsf{Ctx}$$

$$c\colon \mathsf{Ctx} * ((d\colon \mathsf{Ctx} * n\colon \mathbf{N}), a\colon \mathsf{Ty}) \vdash c \circ (d[n\colon a]) = (c \circ d)[n\colon a] : \mathsf{Ctx}$$

With the definitions of object-level terms, types and contexts, we can formalise the Curry-style typing judgement for simply-typed λ-calculus. Informally, this typing judgement is given by the three rules below, which are subject to the usual side-condition on variable names.

$$\frac{}{\Gamma, n\colon a, \Delta \vdash n : a} \qquad \frac{\Gamma \vdash s : a \Rightarrow b \qquad \Gamma \vdash t : a}{\Gamma \vdash \mathsf{app}\ s\ t : b} \qquad \frac{\Gamma, n\colon a \vdash t @_{\mathrm{H}} n : b}{\Gamma \vdash \mathsf{lam}\ t : a \Rightarrow b}$$

These rules for the typing judgement for simply-typed λ-calculus are captured by the following inductive type (HT stands for 'Has Type').

$$c\colon \mathsf{Ctx}, x\colon \mathsf{Lam}, t\colon \mathsf{Ty} \vdash \mathsf{HT}(c,x,t)\ \mathrm{Type}$$

There is one constructor for each typing rule:

$$\vdash \mathsf{htproj} : \Pi c\colon \mathsf{Ctx}.\, \mathrm{H} n.\, \Pi a\colon \mathsf{Ty}.\, \Pi^* d\colon \mathsf{Ctx}.\, \mathsf{HT}(c[n : a] \circ d, \mathsf{var}\ n, a)$$

$$\vdash \mathsf{htapp} : \Pi c\colon \mathsf{Ctx}.\, \Pi s,t\colon \mathsf{Lam}.\, \Pi a,b\colon \mathsf{Ty}.\, \mathsf{HT}(c,s,a \Rightarrow b) \to \mathsf{HT}(c,t,a) \to \mathsf{HT}(c, \mathsf{app}\ s\ t, b)$$

$$\vdash \mathsf{htlam} : \Pi c\colon \mathsf{Ctx}.\, \Pi t\colon (\mathrm{H} n.\, \mathsf{Lam}).\, \Pi a,b\colon \mathsf{Ty}.\, \mathrm{H} n.\, \mathsf{HT}(c[n : a], t @ n, b) \to \mathsf{HT}(c, \mathsf{lam}\ t, a \Rightarrow b)$$

The resulting induction principle is:

$$\frac{\begin{array}{l}\Gamma, c\colon \mathsf{Ctx}, t\colon \mathsf{Lam}, a\colon \mathsf{Ty} \vdash A(c,t,a)\ \mathrm{Type} \\ \Gamma \vdash M : \Pi c\colon \mathsf{Ctx}.\, \mathrm{H} n.\, \Pi a\colon \mathsf{Ty}.\, \Pi^* d\colon \mathsf{Ctx}.\ A(c[n\colon a] \circ d, \mathsf{var}\ n, a) \\ \Gamma \vdash N : \Pi c\colon \mathsf{Ctx}.\, \Pi s,t\colon \mathsf{Lam}.\, \Pi a,b\colon \mathsf{Ty}.\ A(c,s,a \Rightarrow b) \to A(c,t,a) \to A(c, \mathsf{app}\ s\ t, b) \\ \Gamma \vdash R : \Pi c\colon \mathsf{Ctx}.\, \Pi t\colon (\mathrm{H} n.\, \mathsf{Lam}).\, \Pi a,b\colon \mathsf{Ty}.\, \mathrm{H} n.\ A(c[n\colon a], t @ n, b) \to A(c, \mathsf{lam}\ t, a \Rightarrow b)\end{array}}{\Gamma \vdash \mathsf{rec}_A(M,N,R) : \Pi c\colon \mathsf{Ctx}.\, \Pi t\colon \mathsf{Lam}.\, \Pi a\colon \mathsf{Ty}.\, \mathsf{HT}(c,t,a) \to A(c,t,a)}$$

This induction principle is slightly less general than possible, because we have omitted possible HT-assumptions for brevity.

12.2.2 Evaluation

To give an example of how this induction principle can be used, we define the evaluation function for well-typed terms. To this end, we first define a decoding of object-level types, i.e. elements of the inductive type Ty, to actual meta-level types.

$$a\colon \mathsf{Ty} \vdash \mathsf{El}(a)\ \mathrm{Type}$$

$$a\colon \mathsf{Ty}, b\colon \mathsf{Ty} \vdash \mathsf{El}(a \Rightarrow b) \ = \ \mathsf{El}(a) \to \mathsf{El}(b)\ \mathrm{Type}$$

For our purposes, we do not need an equation for $\mathsf{El}(\iota)$, the decoding of the type of individuals, since our object-level λ-calculus has no operations specific to individuals. The universe $\mathsf{El}(x)$ has the evident interpretation in the Schanuel topos. We extend the definition of El to contexts. In a slight abuse of notation, we use the constant El both for types and contexts.

$$c\colon \mathsf{Ctx} \vdash \mathsf{El}(c)\ \mathrm{Type}$$
$$\vdash \mathsf{El}(\mathsf{empty}) = 1\ \mathrm{Type}$$
$$(c\colon \mathsf{Ctx} * n\colon \mathbf{N}), a\colon \mathsf{Ty} \vdash \mathsf{El}(c[n\colon a]) = \mathsf{El}(c) \times \mathsf{El}(a)\ \mathrm{Type}$$

We define an evaluation function for well-typed terms

$$\vdash \mathsf{eval} : \Pi c\colon \mathsf{Ctx}.\, \Pi t\colon \mathsf{Lam}.\, \Pi a\colon \mathsf{Ty}.\, \Pi x\colon \mathsf{HT}(c,t,a).\ \mathsf{El}(c) \to \mathsf{El}(a)$$

by induction on the typing statement $\mathsf{HT}(c,t,a)$.

The most complicated case is the base case. We need to define an appropriate projection function $\mathsf{El}(c[n\colon a] \circ d) \to \mathsf{El}(a)$. Note that the domain of this type is (up to associativity) definitionally equal to $\mathsf{El}(c) \times \mathsf{El}(a) \times \mathsf{El}(d)$. We define the following projection term by induction on d.

$$\vdash P : \Pi d\colon \mathsf{Ctx}.\, \Pi^* n\colon \mathbf{N}.\, \Pi^* c\colon \mathsf{Ctx}.\, \Pi a\colon \mathsf{Ty}.\, \mathsf{El}(c[n\colon a] \circ d) \to \mathsf{El}(a)$$

The base case of this inner induction on d is a term of the following type.

$$\vdash P_{\mathsf{empty}} : \Pi^* n\colon \mathbf{N}.\, \Pi^* c\colon \mathsf{Ctx}.\, \Pi a\colon \mathsf{Ty}.\, \mathsf{El}(c[n\colon a] \circ \mathsf{empty}) \to \mathsf{El}(a)$$

Since we have the definitional equalities $c[n\colon a] \circ \mathsf{empty} = c[n\colon a]$ and $\mathsf{El}(c[n\colon a]) = \mathsf{El}(c) \times \mathsf{El}(a)$, type conversion allows us to make the following definition.

$$P_{\mathsf{empty}} \stackrel{\mathrm{def}}{=} \lambda^* n\colon \mathbf{N}.\, \lambda^* c\colon \mathsf{Ctx}.\, \lambda a\colon \mathsf{Ty}.\, \lambda p\colon \mathsf{El}(c) \times \mathsf{El}(a).\, \mathrm{snd}(p)$$

For the recursion step of the inner induction we must define a term P_{snoc} of the following type

$$\vdash P_{\mathsf{snoc}} : \Pi d\colon \mathsf{Ctx}.\, \Pi h\colon A(d).\, \Pi^* m\colon \mathbf{N}.\, \Pi b\colon \mathsf{Ty}.\, A(d[m\colon b]),$$

where we use the abbreviation

$$A(d) \stackrel{\mathrm{def}}{=} \Pi^* n\colon \mathbf{N}.\, \Pi^* c\colon \mathsf{Ctx}.\, \Pi a\colon \mathsf{Ty}.\, \mathsf{El}(c[n\colon a] \circ d) \to \mathsf{El}(a).$$

Now, we have the definitional equality

$$\mathsf{El}(c[n\colon a] \circ (d[m\colon b])) = \mathsf{El}((c[n\colon a] \circ d)[m\colon b]) = \mathsf{El}(c[n\colon a] \circ d) \times \mathsf{El}(b).$$

Therefore, P_{snoc} must return an element of type $\mathsf{El}(c[n\colon a] \circ d) \times \mathsf{El}(b) \to \mathsf{El}(a)$. The induction hypothesis h provides a function $\mathsf{El}(c[n\colon a] \circ d) \to \mathsf{El}(a)$. Hence, we define P_{snoc} so that it precomposes the function from the induction hypothesis with the first projection. The term is:

$$P_{\mathsf{snoc}} = \lambda d\colon \mathsf{Ctx}.\, \lambda h\colon A(d).\, \lambda^* m\colon \mathbf{N}.\, \lambda b\colon \mathsf{Ty}.\, \lambda^* n\colon \mathbf{N}.\, \lambda^* c\colon \mathsf{Ctx}.\, \lambda a\colon \mathsf{Ty}.\, \lambda x\colon \mathsf{El}(c[n\colon a] \circ d) \times \mathsf{El}(b).$$
$$(((h@n)@c)\ a)\ (\mathrm{fst}(x))$$

With the definitions of P_{empty} and P_{snoc}, the term P is defined by applying the recursion principle for Ctx to P_{empty} and P_{snoc}.

$$\vdash P \stackrel{\text{def}}{=} \mathsf{rec}(P_{\mathsf{empty}}, P_{\mathsf{snoc}}) : \Pi d\colon \mathsf{Ctx}.\, \Pi^* n\colon \mathbf{N}.\, \Pi^* c\colon \mathsf{Ctx}.\, \Pi a\colon \mathsf{Ty}.\, \mathsf{El}(c[n\colon a] \circ d) \to \mathsf{El}(a)$$

To complete the definition of the term defining the base case for eval, we have

$$\dfrac{\dfrac{\dfrac{\dfrac{((d\colon \mathsf{Ctx} * n\colon \mathbf{N}) * c\colon \mathsf{Ctx}), b\colon \mathsf{Ty} \vdash (((P\ d)@n)@c)\ b : \mathsf{El}(c[n\colon b] \circ d) \to \mathsf{El}(b)}{((c\colon \mathsf{Ctx} * n\colon \mathbf{N}) * d\colon \mathsf{Ctx}), b\colon \mathsf{Ty} \vdash (((P\ d)@n)@c)\ b : \mathsf{El}(c[n\colon b] \circ d) \to \mathsf{El}(b)}}{(((c\colon \mathsf{Ctx} * n\colon \mathbf{N}), a\colon \mathsf{Ty}) * d\colon \mathsf{Ctx}), b\colon \mathsf{Ty} \vdash (((P\ d)@n)@c)\ b : \mathsf{El}(c[n\colon b] \circ d) \to \mathsf{El}(b)}}{((c\colon \mathsf{Ctx} * n\colon \mathbf{N}), a\colon \mathsf{Ty}) * d\colon \mathsf{Ctx} \vdash (((P\ d)@n)@c)\ a : \mathsf{El}(c[n\colon a] \circ d) \to \mathsf{El}(a)}$$

Therefore, if we define

$$M \stackrel{\text{def}}{=} \lambda c\colon \mathsf{Ctx}.\, \lambda^{\mathrm{H}} n.\, \lambda a\colon \mathsf{Ty}.\, \lambda^* d\colon \mathsf{Ctx}.\, (((P\ d)@n)@c)\ a$$

then we have

$$\vdash M\ :\ \Pi c\colon \mathsf{Ctx}.\, \mathrm{H} n.\, \Pi a\colon \mathsf{Ty}.\, \Pi^* d\colon \mathsf{Ctx}.\, \mathsf{El}(c[n\colon a] \circ d) \to \mathsf{El}(a).$$

We take this term as the base case of the definition of eval.

In the case for application, the induction hypothesis lets us assume terms of type $(\mathsf{El}(c) \to \mathsf{El}(a \Rightarrow b))$ and $(\mathsf{El}(c) \to \mathsf{El}(a))$, and we have to define a term of type $(\mathsf{El}(c) \to \mathsf{El}(b))$. Because of the definitional equality $\mathsf{El}(a \Rightarrow b)) = (\mathsf{El}(a) \to \mathsf{El}(b))$, we can use meta-level application to define the required term. Hence, we define the term

$$\Gamma \vdash N\ :\ \Pi c\colon \mathsf{Ctx}.\, \Pi s,t\colon \mathsf{Lam}.\, \Pi a,b\colon \mathsf{Ty}.\ (\mathsf{El}(c) \to \mathsf{El}(a \Rightarrow b)) \to (\mathsf{El}(c) \to \mathsf{El}(a)) \to (\mathsf{El}(c) \to \mathsf{El}(b))$$

for the application case by

$$N \stackrel{\text{def}}{=} \lambda c\colon \mathsf{Ctx}.\, \lambda s,t\colon \mathsf{Lam}.\, \lambda a,b\colon \mathsf{Ty}.\, \lambda f\colon (\mathsf{El}(c) \to \mathsf{El}(a \Rightarrow b)).\, \lambda g\colon (\mathsf{El}(c) \to \mathsf{El}(a)).\, \lambda x\colon \mathsf{El}(c).\, (f\ c)\ (g\ c).$$

In the case for λ-abstraction, the induction hypothesis lets us assume $(\mathsf{El}(c[n\colon a]) \to \mathsf{El}(b))$ and we have to define a term of type $(\mathsf{El}(c) \to \mathsf{El}(a \Rightarrow b))$. Using the definitional equalities $\mathsf{El}(c[n\colon a]) = (\mathsf{El}(c) \times \mathsf{El}(a))$ and $\mathsf{El}(a \Rightarrow b)) = (\mathsf{El}(a) \to \mathsf{El}(b))$, we can use meta-level λ-abstraction for this definition. Hence, we define the term

$$\Gamma \vdash R : \Pi c\colon \mathsf{Ctx}.\, \Pi t\colon (\mathrm{H} n.\, \mathsf{Lam}).\, \Pi a,b\colon \mathsf{Ty}.\, \mathrm{H} n.\ (\mathsf{El}(c) \times \mathsf{El}(a) \to \mathsf{El}(b)) \to (\mathsf{El}(c) \to (\mathsf{El}(a \Rightarrow b)))$$

for the λ-binder case by

$$R \stackrel{\text{def}}{=} \lambda c\colon \mathsf{Ctx}.\, \lambda t\colon (\mathrm{H} n.\, \mathsf{Lam}).\, \lambda a,b\colon \mathsf{Ty}.\, \lambda^{\mathrm{H}} n.\, \lambda f\colon (\mathsf{El}(c) \times \mathsf{El}(a) \to \mathsf{El}(b)).\, \lambda x\colon \mathsf{El}(c).\, \lambda y\colon \mathsf{El}(a).\, f\ \langle x, y \rangle.$$

Having defined the terms M, N and R, we define the evaluation function using recursion.

$$\vdash \mathsf{eval} \stackrel{\text{def}}{=} \mathsf{rec}(M, N, R) : \Pi c\colon \mathsf{Ctx}.\, \Pi t\colon \mathsf{Lam}.\, \Pi a\colon \mathsf{Ty}.\, \mathsf{HT}(c, t, a) \to (\mathsf{El}(c) \to \mathsf{El}(a))$$

It can be seen that, semantically, this implements the intended evaluation function.

12.2.3 Limitations

We have shown that simple properties of simply typed λ-calculus, such as the typing judgement and evaluation, can be encoded as types in the type theory. It is natural to ask how much of the standard reasoning about the λ-calculus can be encoded in a propositions-as-types fashion? Unfortunately, the current restriction to closed freefrom types severely restricts such propositions-as-types reasoning.

There is a simple reason that makes propositions-as-types reasoning harder than, say, reasoning in Nominal Logic or the Theory of Contexts. In Nominal Logic or the Theory of Contexts, one can always choose an arbitrary fresh name. Since in both logics any proposition that is true for some fresh name is also true for any other fresh name, a fresh name can be chosen arbitrarily without affecting the truth of the current proposition. In contrast, in type theory the particular choice of a fresh name may well have an effect, for example in $(\text{new } n.n\colon \mathbf{N})$. Hence, in type theory we must *prove* that the choice of a fresh name does not matter. Since we use freefrom types for this purpose, the restriction to closed freefrom types limits our ability to pick fresh names in propositions-as-types reasoning. In effect, we can generate fresh names only for closed propositions, since if A is a type representing a proposition then $(\text{new } n.M\colon A)$ can only be formed if M is of type $A^{*(n:\mathbf{N})}$, and this freefrom type is only available if A is closed.

Furthermore, the restriction to closed freefrom types limits the use of Σ^* as a propositions-as-types existential quantifier expressing 'for some fresh $x\colon A$'. This is the case because the introduction rule for Σ^*-types allows us to form elements of $\Sigma^* x\colon A.B$ only if $\Sigma^* x\colon A.B$ is closed.

To give a concrete example of the restrictions, let us consider the type $\mathsf{HT}(c,t,a)$. In the inductive definition of this type we have formalised the following introduction rule for functions.

$$\frac{\Gamma, n\colon a \vdash t@_{\mathrm{H}} n : b}{\Gamma \vdash \mathsf{lam}\, t : a \Rightarrow b}$$

This rule assumes the judgement $\Gamma, n\colon a \vdash t@_{\mathrm{H}} n : b$ for some fresh name n. As shown by McKinna & Pollack [69], it should be equivalent to formalise the following rule, which makes the same assumption for all fresh names.

$$\frac{\forall^* n\colon \mathbf{N}.\,(\Gamma, n\colon a \vdash t@_{\mathrm{H}} n : b)}{\Gamma \vdash \mathsf{lam}\, t : a \Rightarrow b}$$

This rule can be formalised by replacing the constructor htlam with the following constructor htlam'.

$$\vdash \mathsf{htlam}' : \Pi c\colon \mathsf{Ctx}.\,\Pi t\colon (\mathrm{H}n.\mathsf{Lam}).\,\Pi a,b\colon \mathsf{Ty}.\,(\mathrm{H}n.\mathsf{HT}(c[n:a],t@n,b)) \to \mathsf{HT}(c,\mathsf{lam}\, t, a \Rightarrow b)$$

We would expect both definitions to be provably equivalent, but a derivation of their equivalence requires open freefrom types. Suppose we have an element x of type

$$\mathrm{H}n.\mathsf{HT}(c[n:a],t@n,b) \to \mathsf{HT}(c,\mathsf{lam}\, t, a \Rightarrow b),$$

and we want to define an element y of type

$$(\mathrm{H}n.\mathsf{HT}(c[n:a],t@n,b)) \to \mathsf{HT}(c,\mathsf{lam}\, t, a \Rightarrow b).$$

Intuitively, this element y should be given by $(\lambda z\colon \mathsf{HT}(c[n:a],t@n,b).\,\mathrm{new}\,n.\,(x@n)\;z)$. This term, however, is not typeable since we must show that the choice of n does not matter, i.e. that the subterm $((x@n)\;z)$ is of type $\mathsf{HT}(c,\mathsf{lam}\,t,a\Rightarrow b)^{*(n:\mathbf{N})}$. Even to form this type we need open freefrom types. The other direction also requires open freefrom types, this time because we need to use binding on terms of open type. With an open version of the rule (H-I-Σ^*), we can define x from y by letting x be $\lambda^{\mathrm{H}}n.\,\lambda z\colon \mathsf{HT}(c[n:a],t@n,b).\,y\;(n.z)$.

It is possible to side-step some of the issues that we have just sketched. The problem that with closed freefrom types we cannot obtain sufficient freshness information for the effect-free choice of fresh names can be addressed even without a good formulation of open freefrom types. The main problem in giving a formulation of open freefrom types is to find a good elimination rule. Even if we cannot find a good elimination rule, it is still possible to assume suitable constants that provide sufficient freshness information for practical purposes. In the definition of y from x above, we must be able to find an element of $\mathsf{HT}(c,t,a)^{*(n:\mathbf{N})}$ for any fresh name n and any element of $\mathsf{HT}(c,t,a)$. This amounts to finding a term

$$\mathsf{pure}\colon \mathrm{H}n.\,\mathsf{HT}(c,t,a)\to\mathsf{HT}(c,t,a)^{*(n:\mathbf{N})}.$$

The term pure has an interpretation in the Schanuel topos. It expresses that the elements of $\mathsf{HT}(c,t,a)$ contain no more names than the terms c, t and a together. We call a type for which the Schanuel topos models such a term a *pure* type. The pure types include all propositions, that is types interpreted by a monomorphism in the codomain fibration on $\mathbb{S}$. The fact that all propositions are pure is the reason why names can be chosen without restriction in Nominal Logic. We believe that it is possible to add to the type theory a set of pure-constants that identifies a useful range of pure types. Although not all inductive types are pure—the type Lam, for example, is not—it should nevertheless be possible to give sufficient syntactic criteria on the constructors of an inductive type to infer purity. For example, the constructor htlam' defines a pure type because is takes arguments $c\colon \mathsf{Ctx}$, $t\colon \mathrm{H}n.\,\mathsf{Lam}$, $a,b\colon \mathsf{Ty}$ and $x\colon (\mathrm{H}n.\,\mathsf{HT}(c[n:a],t@_{\mathrm{H}}n,b))$ and the support of these arguments is contained in the support of c,t,a and b, which are just the free variables of the result type $\mathsf{HT}(c,t,a\Rightarrow b)$. Note that the hidden-name quantifier in the type of x removes n from the support. Similar simple arguments work for the other constructors of HT. Hence, just by looking at the inductive definition of HT we can see that it is sound to assume the term pure, by means of which we can side-step some problems with closed freefrom types.

Of course, much better than giving criteria for the semantic existence of the term pure would be to derive it. The natural way of defining pure is by induction on its second argument. A problem with doing this is the fact that the variable n in the type of pure is quantified by a hidden-name type H. Because of this, we cannot directly use the induction principle, which would have a conclusion of type $\Pi c\colon \mathsf{Ctx}.\,\Pi t\colon \mathsf{Lam}.\,\Pi a\colon \mathsf{Ty}.\,\mathrm{H}n.\,\mathsf{HT}(c,t,a)\to\mathsf{HT}(c,t,a)^{*(n:\mathbf{N})}$. If n were quantified by a normal dependent product, then we could just exchange it with c, t and a and use the induction principle. Since n is quantified by H and thus assumed to be fresh for c, t and a, this commonly used exchange trick is not available. Instead, the only way of deriving the term pure by induction appears to be to

transform pure into a function of type (in informal notation) $\Pi c\colon \mathsf{Ctx}.\,\Pi t\colon \mathsf{Lam}.\,\Pi a\colon \mathsf{Ty}.\,\mathsf{HT}(c,t,a) \to \Pi n\colon \mathbf{N}^{*(\langle c,t,a\rangle:\mathsf{Ctx}\times\mathsf{Lam}\times\mathsf{Ty})}.\,\mathsf{HT}(c,t,a)^{*(n:\mathbf{N})}$ for which the induction principle can be used. However, such an inductive definition clearly requires non-trivial manipulation of open freefrom types.

The issue that there are different induction principles whose equivalence would be very desirable but which we cannot show also appears for the type Lam. In Section 12.1.1 we have shown that for closed types we can derive a FM-style induction principle from the normal induction principle. If we want to generalise this to open types, the some/any nature of H-types makes us expect three induction principles with the following respective λ-binder cases:

$$\Gamma \vdash R_1 : \Pi x\colon (\mathrm{H}n.\,\mathsf{Lam}).\,\Pi h\colon (\mathrm{H}n.A(x@n)).A(\mathsf{lam}\ x)$$
$$\Gamma \vdash R_2 : \Pi x\colon (\mathrm{H}n.\,\mathsf{Lam}).\,\mathrm{H}n.A(x@n) \to \big(A(\mathsf{lam}\ x)^{*(n:\mathbf{N})}\big)$$
$$\Gamma \vdash R_3 : \mathrm{H}n.\,\Pi x\colon \mathsf{Lam}.A(x) \to \big(A(\mathsf{lam}\ (n.x))^{*(n:\mathbf{N})}\big)$$

The equivalence of the first and the second term can be seen just as for HT above. The second and the third term should be equivalent since H is part of an equivalence and therefore distributes over products. To show the equivalence of R_2 and R_3, it suffices to show an isomorphism

$$\Gamma \vdash \big(\mathrm{H}n.\,(\Pi x\colon A.B)\big) \cong \big(\Pi x\colon (\mathrm{H}n.A).\,(\mathrm{H}n.B[x@_{\mathrm{H}}n/x])\big)\ \mathrm{Type}, \tag{12.1}$$

because we have $n.(x@_{\mathrm{H}}n) = x$. We can derive the direction from left to right without the need for open freefrom types. It is given by the term

$$\lambda y\colon (\mathrm{H}n.\,(\Pi x\colon A.B)).\,\lambda x\colon (\mathrm{H}n.A).\,\lambda^{\mathrm{H}}n.\,(y@_{\mathrm{H}}n)\ (x@_{\mathrm{H}}n).$$

In the other direction however, we need the elimination for open freefrom types. This direction is given by the following term, in which we abbreviate the type $\Pi x\colon (\mathrm{H}n.A).\,(\mathrm{H}n.B[x@_{\mathrm{H}}n/x])$ by C.

$$\lambda y\colon C.\,\lambda^{\mathrm{H}}n.\,\lambda x\colon A.\,\mathrm{let\ bind}^{\mathrm{H}}(n,x)\ \mathrm{be}\ z^{*n}\colon (\mathrm{H}n.A)^{*(n:\mathbf{N})}\ \mathrm{in}\ (y\ z)@_{\mathrm{H}}n$$

Showing that these two terms do indeed define an isomorphism again requires non-trivial manipulation of open freefrom types. Nevertheless, we should point out that for sufficiently closed types A and B the isomorphism (12.1) is derivable.

It is already evident from the work of McKinna and Pollack [69], that being able to show the equivalence of these different induction principles is important for applications. The practical difference between the different induction principles can be seen by observing that if we had used the definition of HT with htlam′ instead of htlam, then we could not have defined the evaluation function in the previous section. The reason for this is that in the induction case for the λ-binder, we would have had to define a term of type

$$\big(\mathrm{H}n.\,\mathsf{HT}(c[n\colon a],t@_{\mathrm{H}}n,b)\big) \to (\mathsf{El}(c) \to \mathsf{El}(a))$$

and the only way of doing so would have been to define an element of

$$\mathsf{HT}(c[n\colon a],t@_{\mathrm{H}}n,b) \to \big((\mathsf{El}(c) \to \mathsf{El}(a))^{*(n:\mathbf{N})}\big)$$

and choose the name n freshly. For this we would need open freefrom types; and we cannot assume a purity constant either, since the type $\mathsf{El}(c) \to \mathsf{El}(a)$ is not pure. Indeed, the freshness of n from the element of $(\mathsf{El}(c) \to \mathsf{El}(a))$ is not trivial. It would have to be established as part of the inductive definition. In contrast, with the inductive definition of HT using htlam, we do not have choose fresh names. However, the problem of finding fresh does of course not just disappear. It is simply shifted into the definition of HT. Whereas with htlam′ we can construct an element of HT without supplying any name, we do need to supply a name to construct an element of HT using htlam. In other words, the names required for the definition of eval are already contained in the element of HT. This illustrates again that showing the equivalence of the two induction principles comes down to being able to generate fresh names.

Another way of dealing with the problem of fresh name generation is by encoding in the type theory a name-allocation monad, as used in [103], [100] and [21]. Such a monad can be implemented as the type $T(A) \stackrel{\text{def}}{=} \Sigma k\colon \mathrm{Nat}.\, \mathbf{N}^{*k} \multimap A$, where $\mathbf{N}^{*0} = 1$ and $\mathbf{N}^{*k+1} = \mathbf{N} * \mathbf{N}^{*k}$. The operations on T can be defined as described in [103]. However, having to carry around such a monad is likely to be awkward.

12.3 Conclusion

The examples in this and the previous chapter show that even with closed freefrom types interesting examples with names can be defined in the type theory. The examples are comparable to those defined in the modal type theory [98]. However, the examples also show that for serious applications, a generalisation to open freefrom types is necessary. Since, effectively, all the restrictions we have encountered were due to the absence of open freefrom types, there is reason to hope that lifting this restriction will be enough to enable full-scale applications.

Chapter 13

Conclusions and Further Work

13.1 Conclusions

In this thesis we have developed a dependent type theory with names and name-binding. Its main new features for programming and reasoning with names and binding are simple monoidal sums Σ^*, simple monoidal products Π^* and their common special case of hidden-name types H. As a semantic basis for this type theory we have developed a categorical formulation of name-binding. In this categorical formulation, Σ^*, Π^* and H arise naturally from an equivalence of fibrations that may be understood as a formalisation of Barendregt's variable convention. The categorical structure captures directly a number of important FM constructions for working with names and binding. Based on the categorical structure, we have defined a type theory that is in propositions-as-types correspondence to Nominal Logic. We have seen that its definition is not just a matter of appealing to the Curry-Howard isomorphism. The new concept of freefrom types, to which Σ^* and Π^* are intimately related, appears

The syntactic formulation of freefrom types is one of the main limitations of the type theory presented in this thesis. In the examples we have frequently encountered the limitations of our current formulation of closed freefrom types. But we have also seen that in each of these cases the limitation is due only to the syntactic restriction that freefrom types be closed. With open freefrom types, which we have explained categorically, each of these problems can be addressed. On the other hand, even with the restriction to closed freefrom types, our type theory is similar in expressivity to the modal type theory of [98]. Indeed, the complications in generalising closed to open freefrom types appear comparable to the complications in generalising modal type theory to dependent types [76]. We see our precise categorical formulation of freefrom types as a first step towards the solution of the problems with them.

Nevertheless, we do believe the development in this thesis makes a case that Σ^*, Π^* and H are good concepts for integrating names in type theory. Their categorical definition can be seen as a formalisation of Barendregt's variable convention, they are the essence of a number of important constructions in FM theory, and their use for syntax integrates both WHOAS-style and FM-style work.

The categorical definition of H by an equivalence of fibrations not only formalises naturally the informal practice of using interchangeably α-equivalence classes and their freshly named instances, it also characterises directly a number of important constructions for working with names and binding. The direct characterisation allows us to focus on the essential properties of the constructions for names when designing a type theory. This is desirable since constructing the structure for names and binding concretely can be a laborious task, as the length of Chapter 3 illustrates. The categorical definition of this structure allows us to abstract away these details and concentrate on its intrinsic properties. Furthermore, due to the increased generality, we can transfer our work with names and binding to other contexts, such as the context of realizability.

To summarise, the main contributions of this thesis are:

- The definition of bunched dependent type theory with novel multiplicative products Π^*, freefrom types and multiplicative sums Σ^*.
- A categorical semantics for the type theory, including soundness for the interpretation in a class of fibrations and completeness for the fragment of the type theory with Π^* and $*$-types.
- A categorical formulation of FM-style constructions for names and name-binding in terms of an equivalence of fibrations. The construction of concrete models of this structure: the Schanuel topos, a realizability version of the Schanuel topos and the species of structures.
- A type theory with names and hidden-name types using which one can work with syntax both in WHOAS-style and FM-style at the same time.

We believe the work reported in this thesis can serve as a firm foundation for further work on type theory with names and binding.

13.2 Further Work

The work reported here represents the first steps towards a dependent type theory with names and binding—much remains to be done. In addition to the many specific directions for further work discussed in the respective chapters, we mention some directions for this work as a whole.

We have left open many meta-theoretical questions about the type theory, the most important being whether type-checking and definitional equality are decidable. This is an interesting question, especially with the interaction equations of H-types. While we believe it to be essentially straightforward to show decidability of type-checking for the fragment with Π^*, $*$ and freefrom-types, going beyond that appears to be harder. To get a better view of the problem, we should look for simplifications of the type theory.

The restriction to closed freefrom types is the main factor restricting the use of the type theory. An important goal for further work is therefore to find a good formulation of open freefrom types. Finding a simple formulation of (an equivalent of) freefrom types is also the key to simplifying the type theory,

since the let-terms associated to freefrom types are one of main factors contributing to the complexity of the type theory.

Even without a good solution to the formulation of open freefrom types, it is possible to side-step the problems with them by using a dependently sorted predicate logic. By this we mean using a separate first- or higher-order logic on top of the type theory, as described in [56]. Such a logic would contain a freshness quantifier $\mathcal{N}$, which is available by Proposition 10.2.5. It would be a variant of Nominal Logic, and its formulation would be similar to the sequent calculus of Cheney [20, 22]. The problem with freefrom types disappears in such a logic. The pullback in Definition 10.2.1.2 implies that for all propositions $\Gamma \vdash \varphi : \mathrm{Prop}$ there is an equivalence $\Gamma * x\colon A \vdash \varphi \dashv\vdash \varphi^{*(x:A)}$, so that freefrom propositions can simply be dropped.

When looking for a real solution to the problems with freefrom types, we should keep in mind that the problems with them may be just an artifact of our formulation using bunches. While bunches were very important for identifying the semantic structure in Chapter 10, it is possible that other formulations lead to a simpler type theory. For instance, bunches are more general than they need to be for their application to names, being based just on a monoidal structure with some properties. Perhaps we should look for a formulation that uses more of the structure available in categories with bindable names. One possibility for such a formulation would be to assume a universe of propositions $\Gamma \vdash \mathrm{Prop}$, modelled by the subobject logic in the Schanuel topos, and to work with freshness propositions $\Gamma \vdash M\#N : \mathrm{Prop}$ instead of bunches. This would render freefrom type superfluous, thus solving the problem and greatly simplifying the type theory. On the other hand, there are likely to be new problems. For instance, in the formulation of the rule (Σ^*-I) one would encounter a version of the slices in [37], which are likely to be problematic. The details remain to be worked out.

Last but not least, to assess the practicality of the type theory, we should try it. To make it feasible to assess how practical the approach is, there needs to be a prototype implementation.

I wish to God these calculations had been executed by steam.
— CHARLES BABBAGE, 1821

Bibliography

[1] M. Abott, Th. Altenkirch, and N. Ghani. Categories of containers. In *Proceedings of Foundations of Software Science and Computation Structures*, 2003.

[2] S. Abramsky, D. Ghica, A. Murawski, L. Ong, and I. Stark. Nominal games and full abstraction for the nu-calculus. In *Proceedings of the Nineteenth Annual IEEE Symposium on Logic in Computer Science*, pages 150–159. IEEE Computer Society Press, 2004.

[3] R. Adams. Decidable equality in a logical framework with sigma kinds. Unpublished Note, 2001.

[4] S.J. Ambler. *First Order Linear Logic in Symmetric Monoidal Closed Categories*. PhD thesis, University of Edinburgh, 1991.

[5] R. Atkey. A λ-calculus for resource separation. In *Proceedings of ICALP04*, volume 3142 of *LNCS*, 2004.

[6] B.E. Aydemir, A. Bohannon, M. Fairbairn, J.N. Foster, B.C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. Mechanized metatheory for the masses: The POPLmark challenge. In *Proceedings of TPHOLs 2005*. Springer-Verlag, 2005. To appear.

[7] A.G. Barber. *Linear Type Theories, Semantics and Action Calculi*. PhD thesis, University of Edinburgh, 1997.

[8] H. Barendregt. Lambda calculi with types. In D.M. Gabbay S. Abramski and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford Univ. Press, 1992.

[9] H.P. Barendregt. *The Lambda Calculus: Its Syntax And Semantics*. Elsevier Science, 1984.

[10] P.N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *Proceedings of CSL'94*. Springer-Verlag, 1994.

[11] P.N. Benton, G.M. Bierman, J.M.E. Hyland, and V.C.V. de Paiva. A term calculus for intuitionistic linear logic. In *Proceedings of TLCA'93*. Springer-Verlag, 1993.

[12] F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial Species and Tree-like Structures*. Cambridge University Press, 1997.

[13] L. Birkedal. Developing theories of types and computability via realizability. *Electronic Notes in Theoretical Computer Science*, 34, 2000.

[14] N.G. De Bruijn. Lambda calculus notation with nameless dummies: A tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[15] L. Cardelli and L. Caires. A spatial logic for concurrency (part I). *Information and Computation*, 186:194–235, 2003.

[16] L. Cardelli, P. Gardner, and G. Ghelli. Manipulating trees with hidden labels. In *Proceedings of FOSSACS'03*, volume 2620 of *LNCS*. Springer-Verlag, 2003.

[17] L. Cardelli and A. Gordon. Logical properties of name restriction. In *Proceedings of TLCA'01*, volume 2044 of *LNCS*. Springer-Verlag, 2001.

[18] J. Cartmell. Generalized algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

[19] I. Cervesato and F. Pfenning. A linear logical framework. *Information and Computation*, 1998.

[20] J.R. Cheney. *Nominal Logic Programming*. PhD thesis, Cornell University, 2004.

[21] J.R. Cheney. Scrap your nameplate (functional pearl). In *Proceedings of ICFP 2005*, 2005. To appear.

[22] J.R. Cheney. A simpler proof theory for nominal logic. In *Proceedings of FOSSACS 2005*, number 3441 in LNCS, pages 379–394. Springer-Verlag, 2005.

[23] J.R. Cheney and C. Urban. Alphaprolog: A logic programming language with names, binding and alpha-equivalence. In *Proceedings of ICLP 2004*, volume 3132 of *LNCS*, pages 269–283. Springer-Verlag, 2004.

[24] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[25] T. Coquand and A. Abel. Untyped algorithmic equality for Martin-Löf's logical framework with surjective pairs. In *Proceedings of TLCA'05*, 2005.

[26] P.L. Curien. Substitution up to isomorphism. *Fundamenta Informaticae*, 19:51–85, 1993.

[27] B.J. Day. On closed categories of functors. In *Reports of the Midwest Category Seminar, IV*, volume 137 of *Lecture Notes in Mathematics*, pages 1–38. Springer-Verlag, 1970.

[28] N.G. de Bruijn. Telescopic mappings in typed lambda-calculus. *Information and Computation*, 91:189–204, 1991.

[29] J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order abstract syntax in Coq. In *Proceedings of TLCA'95*, 1995.

[30] M. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings of LICS99*, 1999.

[31] M. Fiore and D. Turi. Semantics of name and value passing. In *Proceedings of LICS01*, 2001.

[32] D. Fridlender. A proof-irrelevant model of Martin-Löf's logical framework. *Mathematical Structures in Computer Science*, 12:771–795, 2002.

[33] M.J. Gabbay. *A Theory of Inductive Definitions With α-equivalence: Semantics, Implementation, Programming Language*. PhD thesis, University of Cambridge, 2000.

[34] M.J. Gabbay. FM-HOL, a higher-order theory of names. In *Workshop on Thirty Five years of Automath*, 2002.

[35] M.J. Gabbay. Fresh logic. Submitted, 2003.

[36] M.J. Gabbay. A general mathematics of names in syntax. Submitted, March 2004.

[37] M.J. Gabbay and J.R. Cheney. A sequent calculus for nominal logic. In *Proceedings of LICS 2004*, pages 139–148. IEEE Computer Society Press, 2004.

[38] M.J. Gabbay and A.M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.

[39] F. Gadducci, M. Miculan, and U. Montanari. Some characterization results for permutation algebras. In *Proceedings of COMETA03*, 2003.

[40] H. Goguen. *A Typed Operational Semantics for Type Theory*. PhD thesis, University of Edinburgh, 1994.

[41] A. Gordon and T. Melham. Five axioms of alpha-conversion. In J. von Wright, J. Grundy, and J. Harrison, editors, *TPHOLs 1996*, volume 1125 of *LNCS*, pages 173–190. Springer-Verlag, 1996.

[42] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

[43] R. Harper and F. Pfenning. On equivalence and canonical forms in the LF type theory. *Transactions on Computational Logic*, 6:61–101, 2005.

[44] C. Hermida. *Fibrations, Logical Predicates and Indeterminates*. PhD thesis, University of Edinburgh, 1993.

[45] C. Hermida. Some properties of fib as a fibred 2-category. *Journal of Pure and Applied Algebra*, 134(1):83–109, 1999.

[46] M. Hofmann. On the interpretation of type theory in locally cartesian closed categories. In *Proceedings of CSL94*, volume 933 of *LNCS*. Springer-Verlag, 1994.

[47] M. Hofmann. *Extensional Concepts in Intensional Type Theory*. PhD thesis, University of Edinburgh, 1995.

[48] M. Hofmann. Syntax and semantics of dependent types. In P. Dybjer and A.M. Pitts, editors, *Semantics of Logics of Computation*. Cambridge Univ. Press, 1997.

[49] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *Proceedings of LICS99*, 1999.

[50] M. Hofmann and Th. Streicher. The groupoid interpretation of type theory. In *Venice Festschrift*. 1996.

[51] F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning about nominal algebras in HOAS. In *Proceedings of ICALP01*, 2001.

[52] J.M.E. Hyland. The effective topos. In *The L.E.J. Brouwer Centenary Symposium*, volume 110 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1982.

[53] J.M.E. Hyland and A.M. Pitts. The theory of constructions: Categorical semantics and topos-theoretic models. *Contemporary Mathematics*, 92:137–199, 1989.

[54] B. Jacobs. *Categorical Type Theory*. PhD thesis, University of Nijmegen, 1991.

[55] B. Jacobs. Comprehension categories and the semantics of type theory. *Theoretical Computer Science*, 107:169–207, 1993.

[56] B. Jacobs. *Categorical Logic and Type Theory*. Elsevier Science, 1999.

[57] P.T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Oxford University Press, 2002.

[58] A. Joyal. Une théorie combinatoire des séries formelle. *Advances in Mathematics*, 42:1–82, 1981.

[59] G.M. Kelly. Many-variable functorial calculus I. In *Coherence in Categories*, number 281 in Lecture Notes in Mathematics, pages 66–105. Springer-Verlag, 1972.

[60] P. Lietz. A fibrational theory of geometric morphisms. Master's thesis, TU Darmstadt, 1998.

[61] J.R. Longley. *Realizability Toposes and Language Semantics*. PhD thesis, University of Edinburgh, 1994.

[62] J.R. Longley and A.K. Simpson. A uniform approach to domain theory in realizability models. *Mathematical Structures in Computer Science*, 7(5):469–505, 1997.

[63] S. MacLane. *Categories for the Working Mathematician*. Springer-Verlag, 1997.

[64] S. MacLane and I. Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer-Verlag, 1992.

[65] P. Martin-Löf. *Intuitionistic Types Theory*. Bibliopolis, 1984.

[66] C. McBride. *Dependently Typed Functional Programs and their Proofs*. PhD thesis, University of Edinburgh, 1999.

[67] C. McBride and J. McKinna. Functional pearl: I am not a number—I am a free variable. In *ACM SIGPLAN 2004 Haskell Workshop, Snowbird, Utah, USA*, 2004.

[68] R. McDowell and D. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transaction in Computational Logic*, 3(1):80–136, 2002.

[69] J. McKinna and R. Pollack. Some type theory and lambda calculus formalised. *Journal of Automated Reasoning*, 23(3–4):373–409, 1999.

[70] P.A. Melliès. Mac Lane's coherence theorem viewed as a word problem. Technical Report 25, Prépublication de l'équipe PPS, 2003.

[71] M. Menni. *Exact Completions and Toposes*. PhD thesis, University of Edinburgh, 2000.

[72] M. Menni. About И-quantifiers. *Applied Categorical Structures*, 11(5):421–445, 2003.

[73] D. Miller and A. Tiu. A proof theory for generic judgments. *ACM Transactions on Computational Logic*, 2005. To appear.

[74] A. Momigliano and A. Tiu. Induction and co-induction in sequent calculus. In *Proceedings of TYPES 2003*, volume 3085 of *LNCS*, pages 293–308. Springer-Verlag, 2003.

[75] G. Nadathur and D. Miller. Higher-order logic programming. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intel ligence and Logic Programming*, volume 5, pages 499–590. Oxford University Press, 1998.

[76] A. Nanevski, B. Pientka, and F. Pfenning. A modal foundation for meta variables. In *Proceedings of MERλIN 2003*. ACM Press, 2003.

[77] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory*. Oxford University Press, 1990. Available from www.cs.chalmers.se/Cs/Research/Logic/book.

[78] M. Norrish. Mechanising Hankin and Barendregt using the Gordon-Melham axioms. In *Proceedings of MERλIN 2003*, pages 1–7. ACM Press, 2003.

[79] M. Norrish. Recursive function definition fo types with binders. In *Proceedings of TPHOLs 2004*, volume 3223 of *LNCS*, pages 241–256. Springer-Verlag, 2004.

[80] P. O'Hearn. On bunched typing. *Journal of Functional Programming*, 13(4):747–796, 2003.

[81] E. Palmgren and V. Stoltenberg-Hansen. Domain interpretations of Martin-Löf's partial type theory. *Annals of Pure and Applied Logic*, 48:135–196, 1990.

[82] L.C. Paulson. The foundation of a generic theorem prover. *Journal Automated Reasoning*, 5, 1989.

[83] F. Pfenning and H. Wong. On a modal λ-calculus for S4. *Electronic Notes in Theoretical Computer Science*, 1, 1995.

[84] W.K.-S. Phoa. An introduction to fibrations, topos theory, the effective topos and modest sets. Technical Report LFCS-92-208, University of Edinburgh, 1992.

[85] A.M. Pitts. Categorical logic. In S. Abramski, D.M. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 5, chapter 2. Oxford Univ. Press, 2000.

[86] A.M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.

[87] A.M. Pitts and M.J. Gabbay. A metalanguage for programming with bound names modulo renaming. In *Proceedings of MPC2000*, volume 1837 of *LNCS*, pages 230–255. Springer-Verlag, 2000.

[88] R. Pollack. Closure under alpha-conversion. In *Proceedings of TYPES93*, volume 806 of *LNCS*, pages 313–332. Springer-Verlag, 1993.

[89] R. Pollack. Reasoning about languages with binding. Can we do it yet?, February 2005. Slides. Available from http://homepages.inf.ed.ac.uk/rap.

[90] A.J. Power. A unified category-theoretic approach to variable binding. In *Proceedings of MERλIN 2003*. ACM Press, 2003.

[91] D. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*. Kluwer Academic Publishers, 1999.

[92] D. Pym and S. Ishtiaq. A relevant analysis of natural deduction. *Journal of Logic and Computation*, 8(6), 1998.

[93] E. Ritter. *Categorical Abstract Machines for Higher-Order Typed Lambda Calculi*. PhD thesis, University of Cambridge, 1992.

[94] A. Salvesen. Polymorphism and monomorphism in Martin-Löf's type theory. Logic Colloquium'88, 1988.

[95] U. Schöpp and I. Stark. A dependent type theory with names and binding. In *Proceedings of CSL'04*, volume LNCS of *3210*, pages 235–249. Springer-Verlag, 2004.

[96] L. Schröder. The logic of the partial λ-calculus with equality. In *Proceedings of CSL'04*. Springer-Verlag, 2004.

[97] C. Schürmann. *Automating the Meta Theory of Deductive Systems*. PhD thesis, Carnegie Mellon University, 2000.

[98] C. Schürmann, J. Despeyroux, and F. Pfenning. Primitive recursion for higher-order abstract syntax. *Theoretical Computer Science*, 266(1–2):1–57, 2001.

[99] R.A.G. Seely. Locally cartesian closed categories and type theory. In *Math. Proc. Cambridge Philos. Soc.*, volume 95, pages 33–48, 1984.

[100] M.R. Shinwell. *The Fresh Approach: functional programming with names and binders*. PhD thesis, University of Cambridge, 2005.

[101] M.R. Shinwell and A.M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 200X. to appear.

[102] M.R. Shinwell, A.M. Pitts, and M.J. Gabbay. FreshML: Programming with binders made simple. In *Eighth ACM SIGPLAN International Conference on Functional Programming (ICFP 2003), Uppsala, Sweden*, pages 263–274. ACM Press, 2003.

[103] I.D.B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, 1994.

[104] A. Stoughton. Substitution revisited. *Theoretical Computer Science*, 59:317–325, 1988.

[105] Th. Streicher. *Semantics of Type Theory*. Birkhäuser, 1991.

[106] M. Takeyama. *Universal Structure and a Categorical Framework for Type Theory*. PhD thesis, University of Edinburgh, 1995.

[107] M. Tanaka. Abstract syntax and variable binding for linear binders. In *Proceedings of MFCS00*, volume 1893 of *LNCS*. Springer-Verlag, 2000.

[108] M. Tanaka. *Pseudo-Distributive Laws and a Unified Framework for Variable Binding*. PhD thesis, University of Edinburgh, 2004.

[109] A. Tasistro. *Substitution, Record Types and Subtyping in Type Theory, with Applications to the Theory of Programming*. PhD thesis, Chalmers University of Technology, 1997.

[110] P. Taylor. *Practical Foundations of Mathematics*. Cambridge University Press, 1999.

[111] D. Turi. Category theory lecture notes. Available from http://www.dcs.ed.ac.uk/home/dt/CT/, 2001.

[112] C. Urban, A.M. Pitts, and M.J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1–3):473–497, 2004.

[113] C. Urban and C. Tasson. Nominal reasoning techniques in Isabelle/HOL. In *Proceedings of CADE-20*, LNCS. Springer-Verlag, 2005. To appear.

[114] R. Vestergaard. *The primitive proof theory of the λ-calculus*. PhD thesis, Heriot-Watt University, Edinburgh, 2003.

[115] R. Vestergaard. Some/any is easy, effective and necessary. In *Symposium on Mathematical Logic*, 2003.

[116] R. Vestergaard and J. Brotherston. A formalized first-order confluence proof of the λ-calculus using one sorted variable names. *Information and Computation*, 183(2):212–244, 2003.

[117] O. Wyler. *Lecture Notes on Topoi and Quasitopoi*. World Scientific, 1991.

[118] N. Yoshida and M. Hennessy. Assigning types to processes. *Information and Computation*, 173:82–120, 2002.

Index of Rules

Index